Кириченко А.А.

Программное обеспечение нейросетевых исследований.

2016г.

Кириченко А.А.

«Программное обеспечение нейросетевых исследований»

М.: Изд. HSE, 2016. Сетевое электронное издание монографии. 152 страницы, формат PDF.

ISBN 978-5-9904911-6-8

Рецензент: Сергеев Геннадий Александрович, к.т.н.,с.н.с., ген. директор ООО “Электроком-М”

Монография представляет собой результат изучения проблемы нейросетевого исследования. Для проведения такого исследования необходимы нейросетевые пакеты и программные комплексы, различные инструменты (алгоритмы, программы, методики) и технологии. Чаще всего требуемые средства отсутствуют или доступны по запредельным ценам. Относительно несложные инструменты, необходимые для исследования, могут быть изготовлены самостоятельно за счёт комплексирования программных средств, технология которого связана с использованием системы команд ОС Windows, конвейеризацией команд ОС, использованием нетрадиционных возможностей известных программных средств и недокументированных возможностей алгоритмического языка C#, безинтерфейсного построения сложных программных комплексов и управления исследованиями с помощью API и из командной строки.

Книга может быть полезна студентам, магистрам и аспирантам при подготовке и проведении нейросетевых исследований.

Кириченко А.А. профессор департамента программной инженерии факультета компьютерных наук Федерального государственного автономного образовательного учреждения высшего профессионального образования “Национальный исследовательский университет "Высшая школа экономики" ”.

ISBN 978-5-9904911-6-8

Содержание.

Архитектура нейросетевого исследования.

Универсальные ЭВМ неэффективны при решении трудноформализуемых задач – задач, алгоритм решения которых неизвестен. Они неспособны обучаться решению какой-либо задачи «на примерах». А живые организмы такой способностью обладают!

Разница объясняется, видимо, тем, что принципиально отличаются методы решения задач, характерные для живых организмов и ЭВМ.

Одновременно с разработкой методов распознавания, реализуемых на

универсальных ЭВМ фон-Неймановской архитектуры, ведется поиск методов решения задач, основанных на иных принципах.

Наиболее интересные результаты в этом направлении научных исследований получены при попытках создания распределенных самоорганизующихся систем, например, перцептрона Розенблатта, пандемониума Селфриджа, нейронных ЭВМ (сокращенно – нейроЭВМ), моделирующих работу нервных сетей живых организмов.

На их основе в настоящее время разработаны и практически используются параллельные, волновые, матричные и др. системы распознавания.

НейроЭВМ характеризуются:

- принципом действия: это однородная система, состоящая из большого числа очень простых элементов;
- тем, что нейроЭВМ не требуют программирования; нужно обучение на специально подобранных примерах, составляющих обучающую выборку;
- способностью к обучению и самообучению (способность к самообучению начинает появляться при наличии около 100 нейронов);
- способностью решать задачи на примерах, отсутствовавших в обучающей выборке – при этом если нейросистема обучена распознавать образы, то полученный навык она сохраняет при предъявлении ей части объекта или при показе ей объекта, повернутого под другим углом;

НейроЭВМ имеют высокую стоимость. Поэтому предпринимаются попытки реализовать алгоритмы работы нейросети на универсальных ЭВМ. Такие программы называются нейроимитаторами, когнитронами или нейропакетами. Обычно они используются в экспериментальных исследованиях. В последнее время стали появляться программы, позволяющие решать практические задачи и проводить различные исследования с помощью моделей нейросетей.

Для проведения нейросетевого исследования кроме понимания его технологии необходимо иметь подходящий набор инструментов, в состав которых входят различные программы, алгоритмы, технические средства. Все они входят в состав нейросетевого программирования, включающего в себя:

1. Нейросетевые пакеты и программные комплексы.
2. Модели нейросетевых компонентов.
3. Инструменты для проведения нейросетевых исследований
4. Технологию комплексирования программных средств, включающую в себя
    - использование системы команд операционной системы
    - использование недокументированных возможностей C#
    - установку и настройку среды разработки, средств контроля и преобразования
5. Алгоритмические языки. (Здесь приводится Практикум по основам программирования на C#).
6. Алгоритмы, средства и методы интеллектуализации программных средств, включающие в себя
    - Интеллектуальные алгоритмы.
    - Архитектуру интеллектуальных программ.
    - Встраивание в программу внешних пакетов.
    - Нейронные вставки в создаваемую программу.

Технология проведения нейросетевого исследования.

Проведение научного исследования чаще всего заключается в выявлении скрытых правил и закономерностей в наборах данных, формулировке гипотез и выявлении типовых структур (паттернов). Для этого приходится использовать различные методы обнаружения (добычи) знаний: абстрагирование, ассоциативное объединение, классификацию, кластеризацию, анализ временных рядов, прогнозирование и др.

Человеческий разум не приспособлен для восприятия больших массивов разнородной информации. В среднем человек не способен улавливать более двух-трех взаимосвязей даже в небольших выборках.

Для расширения аналитических возможностей человека можно использовать методы традиционной статистики, эвристические решающие устройства на основе экспертных систем, семантический дифференциал, теорию решения изобретательских задач (ТРИЗ), нейронные сети.

Традиционная статистика решает аналогичные задачи, но она оперирует усредненными характеристиками выборки, которые часто являются фиктивными величинами, например - средней платежеспособностью клиента, в то время, как необходимо уметь прогнозировать состоятельность и намерения конкретного клиента с учётом функции риска или функции потерь.

Методы математической статистики, эвристические решающие устройства, семантический дифференциал, ТРИЗ, так же, как и статистика относятся к дискретным методам. Для человека же несвойственно использовать при решении жизненных проблем дискретные методы.

Естественным для человека является использование основных принципов мозга — ассоциативное мышление, использование принципов обучения (самообучения) и адаптации, нечёткой логики и связей «если - то», «посылка - следствие», лежащих в основе распознавания, движения, управления, принятия решений.

Поэтому из различных способов расширения аналитических возможностей человека наиболее эффективными при исследовании задач, не имеющих общепризнанного алгоритма решения, является использование нейронных сетей.

Всё, что связано с использованием нейронных сетей получило название нейросетевых технологий.

Считается, что нейросетевые технологии не требуют программирования, а предусматривают работу по обучению нейронной сети на специально подобранных примерах. Но нужно понимать, что реализация нейросетевых технологий требует наличия либо аппаратных, либо специальных программных средств.

Основной функцией обучения нейросети, воспроизводящей работу мозга и ассоциативное мышление является узнавание, умение определять сходство и различия.

На этапе обучения формируются основные отношения между входными параметрами и оформляются в незримые таблицы (образы), которые впоследствии будут использоваться при решении задач на сети.

Нейросети наиболее приспособлены к решению широкого круга задач, так или иначе связанных с обработкой образов. Вот список типичных математических постановок задач для нейросетей:

1. Узнавание (Классификация)
2. Кластеризация
3. Регрессия (прогнозирование, предсказание)
4. Понижение размерности

В задачах регрессии целью является оценка значения числовой (принимающей непрерывный диапазон значений) выходной переменной по значениям входных переменных.

В задачах анализа временных рядов целью является **прогноз** будущих значений переменной, зависящей от времени, на основе предыдущих значений ее и/или других

переменных.

Как правило, прогнозируемая переменная является числовой, поэтому прогнозирование временных рядов - это частный случай регрессии. Однако такое ограничение часто в пакет не закладывается, так что в нем можно прогнозировать и временные ряды номинальных (т.е. классифицирующих) переменных.

В задаче узнавания сеть должна отнести каждое наблюдение к одному из нескольких классов (или, в более общем случае, оценить вероятность принадлежности наблюдения к каждому из классов).

Задачи классификации (кластеризации) заключаются в группировке похожих образов в классы (кластеры). В нейропакетах они решаются с помощью сети СОКК (самоорганизующаяся карта Кохонена), или так называемой радиальной базисной функцией (radial basis function - RBF)

Самоорганизующаяся карта Кохонена (СОКК или сеть Кохонена) принципиально отличается от всех других типов сетей. В то время как все остальные сети предназначены для задач с управляемым обучением (т.е обучением с учителем), сети Кохонена главным образом рассчитаны на неуправляемое обучение (без учителя).

В алгоритмах неуправляемого обучения значения весов и/или порогов меняются на основании только входных обучающих данных (выходные значения не требуются и, если они присутствуют, игнорируются).

Этот список можно было бы продолжить и дальше. За ними просматривается некий единый прототип, позволяющий при известной доле воображения сводить их друг к другу. Чаще всего - это типичные примеры некорректных задач, т.е. задач не имеющих единственного решения, алгоритмы которых неизвестны или не имеют строго обоснования.

Нейрокомпьютинг предоставляет единую методологию решения очень широкого круга практически интересных задач. Это, как правило, ускоряет и удешевляет разработку приложений.

В число таких задач входят прогнозирование цен, оценка кредитоспособности, оптическое распознавание (например, подписи), обработка изображений, диагностика, лингвистический анализ, и др.

Нейросети - это не что иное, как новый инструмент анализа данных.

И лучше других им может воспользоваться именно специалист в своей предметной области.

Основные трудности на пути еще более широкого распространения нейротехнологий - в неумении широкого круга профессионалов формулировать свои проблемы в терминах, допускающих простое нейросетевое решение, т.е. неумение проводить нейросетевую декомпозицию решаемой задачи.

Поскольку основные задачи, решаемые с помощью нейропакетов, связаны с поиском скрытых закономерностей, классификацией, прогнозированием, сокращением размерности, область их применения охватывает все направления человеческих знаний – и технических, и гуманитарных.

Изучение возможностей нейропакетов, моделирование разнообразных нейронных систем, разработка программ для автоматизации выполнения интеллектуальных операций, настройка и обучение универсальных нейропакетов на решение определённых задач – вот далеко не полный перечень задач, стоящих перед бакалаврами и магистрами национального исследовательского университета.

Нейросетевое исследование предусматривает выполнение следующих этапов:

1. Подготовка исходных данных.
2. Разметка исходных данных.
3. Формирование нейронной сети.
4. Предварительное обучение сети.

5. Анализ результатов обучения.
6. Оптимизация обучения сети.
7. Анализ подготовленных для исследования данных.
8. Проведение нейросетевого исследования.

Подготовка данных для исследования с помощью нейронной сети.

Данные в нейронной сети используются прежде всего – для обучения. С этой целью они оформляются в виде обучающей выборки, или обучающего набора данных.

Например, сеть необходимо обучить классификации на два класса по косвенным признакам или обучить прогнозированию.

Примерами таких задач могут служить следующие:

- «Мужчина/женщина»,
- «Ирисы Фишера»
- «Как выбирают американских президентов»
- и др.

В задаче «Мужчина/женщина» в общем виде обучающий набор данных может представлять собой таблицу вида:

Информационные (inf)	Исходные показатели (Input)				Результирующие (Pattern)
	Готовите ли вы дома пищу	Как часто вы убираете квартиру	Сколько времени в неделю вы тратите на ремонт автомобиля	Является ли только ваш заработок основным источником дохода семьи	
1	нет	редко	3 часа	да	м
2	Да	всегда	0	нет	ж
…	…	…	…	…	…

Разметка исходных данных.

При разметке исходных данных создаются тренировочный, тестовый и экзаменационный наборы данных. Тренировочный набор данных содержит обучающую выборку, он используется для обучения нейронной сети.

Тестирование необходимо для проверки создания обобщённых образов, которые были представлены в обучающей выборке данных.

Для адекватной оценки качества полученной сети используется ещё один совершенно независимый набор данных, называемый обычно экзаменационным. В процессе обучения сети, определения момента остановки обучения, определения дополнительных параметров и др. этот набор вообще никак не используется, т.е. при последующем применении сеть "видит" его впервые. Поэтому качество результатов, показанных сетью на экзаменационном наборе, является хорошей оценкой качества результатов, которые сеть сможет показать на новых, неизвестных ей данных. Разумеется, экзаменационный набор также должен быть представительным.

Программы пакета, предназначенные для решения задач классификации, позволяют задавать все три набора данных, как по отдельности, так и путём установления доли каждого из наборов в общем количестве данных. При применении сети статистика рассчитывается отдельно для каждого класса и каждого набора.

Формирование нейронной сети.

Формирование нейронной сети: определение типа сети, количества слоёв, количества нейронов во входном, выходном и промежуточных слоях, и др.

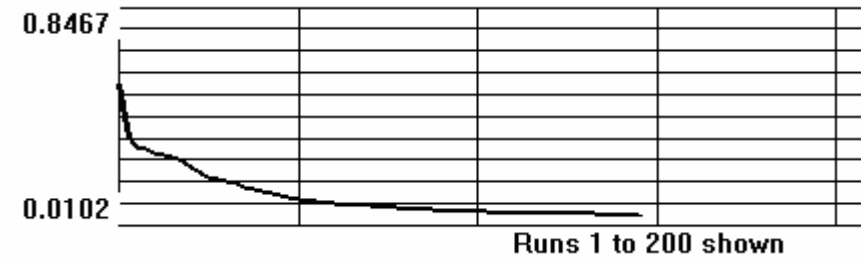
Установка параметров нейронной сети.

Состав параметров определяется в зависимости от используемого вида нейронной сети. Так, для многослойного перцептрона могут быть определены:

- допустимая погрешность обучения (training tolerance);
- допустимая погрешность тестирования (testing tolerance);
- условия остановки обучения:
    1. после окончания N эпохи;
    2. при отсутствии ошибок во время выполнения всей тестирующей выборки;
    3. при достижении M% правильных результатов обучающей выборки;
    4. при уменьшении ошибки сети ниже K%;
- особенности процесса тестирования:
    - тестирование сети после прохождения каждых k эпох;
    - тестирование сети после каждых p обучающих примеров;
- особенности сохранения результатов (например, после завершения каждых r эпох);
- должна ли быть создана и сохранена в файле начальная матрица коэффициентов связи перцепторов с сумматором.
- и т.д.

Предварительное обучение, анализ результатов и оптимизация обучения сети

Проводится предварительное обучение, анализ результатов и оптимизация обучения сети. В процессе обучения сети ей на вход подаётся один из примеров обучающей выборки и фиксируется результат (output) на выходе сети. Поскольку сеть ещё не обучена, этот результат отличается от желаемого (target). Величина отличия называется ошибкой сети. Сначала она имеет большую величину, по мере обучения – уменьшается:

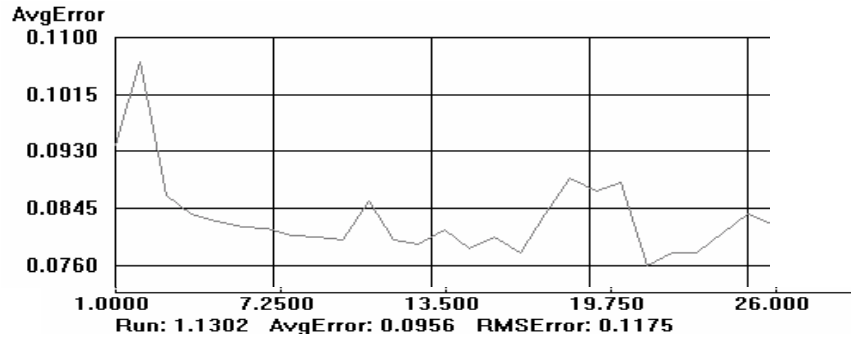


Процесс обучения сети заключается в изменении весов связей перцепторов с сумматором таким образом, чтобы ошибка сети уменьшилась.

Когда весь запас обучающих примеров исчерпан, считается, что закончилась одна эпоха. Подсчитывается средняя ошибка за эпоху и примеры из обучающей выборки снова используются для обучения той же слегка обученной сети – это вторая эпоха.

На приведенном рисунке приведен график, отражающий результаты 200 эпох.

После каждых k эпох обучения фиксируются результаты, и вместо обучающей выборки на вход сети подаются примеры из тестирующей выборки. По ним так же определяется ошибка сети, но обучение сети не проводится. Когда примеры из тестирующей выборки исчерпаны, определяется средняя ошибка тестирования. График изменения ошибки сети при тестировании может быть таким:



Этот график демонстрирует явление, которое называется «переучиванием»: т.е. достижение такого состояния сети, в котором сеть со 100% точностью распознаёт все примеры тренировочного набора данных, однако на других данных, не входивших в этот набор, может давать большую погрешность. Можно сказать, что такая сеть плохо обобщает. Из графика видно, что где-то между 7 и 13 эпохами есть точка, после которой ошибка сети резко возрастает. Это значит, что после эпохи, в которой получена эта точка, продолжать обучение по той же обучающей выборке бесполезно. Прекращение тренировки при минимуме ошибки на тестовом наборе позволяет зафиксировать сеть в том состоянии, когда она уже усвоила наиболее существенную информацию, содержащуюся в данных, и неплохо обобщает, однако ещё не успела переучиться.

Можно уточнить полученную информацию по результатам, сохранённым в файле при обучении. Читаем значения Run, AvgError и RMSError и заносим их в таблицу:

Run	AvgError	RMSError
3,2135	0,0859	0,1019
3,9947	0,0837	0,1008
4,1250	0,0835	0,1009
4,9062	0,0826	0,1019
6,0781	0,0817	0,1018
7,0546	0,0815	0,1030
8,0312	0,0803	0,1031
9,0072	0,0802	0,1033
9,9192	0,0798	0,1034
10,049	0,0800	0,1037
11,026	0,0853	0,1083

Судя по графику AvgError, улучшение работы сети продолжается до 9 эпохи. На 9-10 эпохе средняя ошибка минимальна. Возрастание её в дальнейшем связано с «переучиванием» сети. Желательно обучение сети на 10 эпохе закончить.

Оптимизация обучения сети заключается в том, что полученные результаты обучения стираются, восстанавливается сохраненная в файле начальная матрица коэффициентов связи перцепторов с сумматором, устанавливается ограничение условия остановки обучения: после окончания 10 эпохи, и снова проводится обучение нейронной сети. Заново обученную сеть можно считать оптимальной, она сохраняется в файле и в дальнейшем может быть использована для решения задач данного типа.

Подготовка данных для исследования на обученной сети.

Исследование лучше проводить на заранее подготовленном файле, содержащем исходные данные. Такой файл не должен содержать колонку pattern, результат будет отражаться в колонке out.

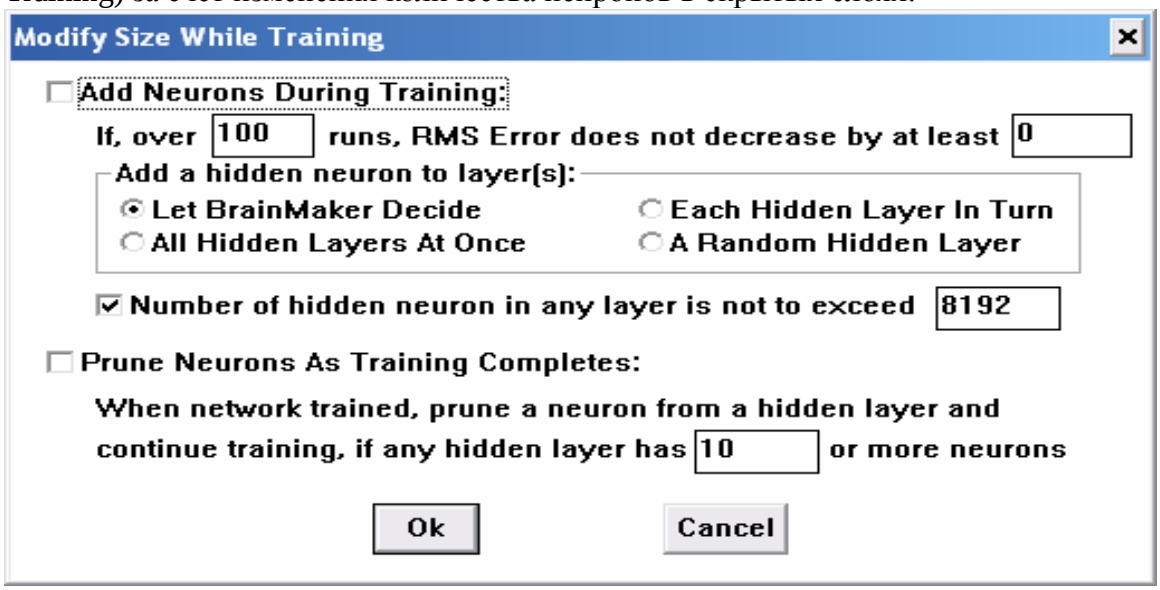
Проведение нейросетевого исследования.

В составе нейросетевых пакетов можно использовать имеющиеся средства для наблюдения за работой (например, за процессом обучения).

a. В пакете Brain Maker для наблюдения за ошибкой распознавания можно

использовать пункты меню Display и Parameters. Пункт “Network Progress Display” в меню “Display” в открывшемся окне позволяет наблюдать, как уменьшается значение ошибки распознавания в процессе обучения.

б. Можно включить модификацию сети во время обучения (Modify Size While Training) за счёт изменения количества нейронов в скрытых слоях:

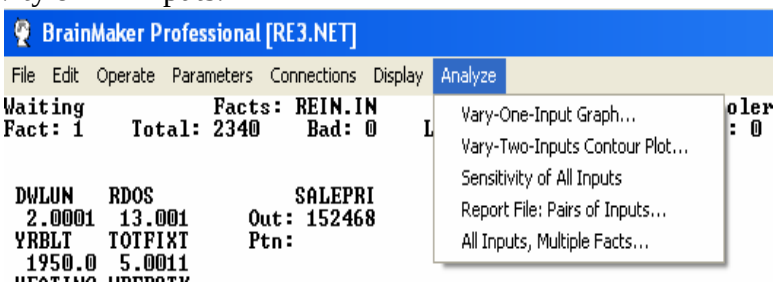


в. Есть ещё одно средство наблюдения - пункт “Show Histograms” в меню “Display” покажет, как нейронная сеть меняет свои внутренние свойства, в частности, теряет, по мере накопления знаний, *способность* обучаться.

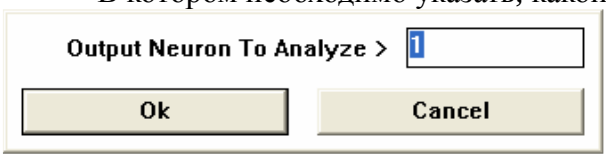
Если после обучения нейросеть эффективно обрабатывает данные обучающего множества, важным становится исследование эффективности работы с данными, которые не использовались для обучения.

В блок анализа нейропакетов обычно входит функция оценки чувствительности для одного входа. Она демонстрирует, насколько данный вход оказывает влияние на выход. Предполагается, что с помощью этой функции можно исследовать поведение нейронной сети с целью выявления мало чувствительных входов или поддиапозонов внутри множества принимаемых входом значений для их последующего исключения в момент проектирования нейронной сети.

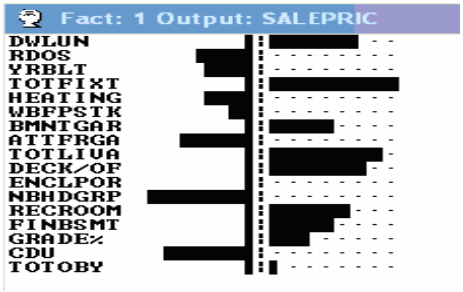
Проверка чувствительности всех входов производится через меню Analuze → Sensitivity of All Inputs.



В котором необходимо указать, какой именно нейрон будет исследоваться:



В пакете BM указывается чувствительность входов с точностью +/-10%. Результат виден на графике (термометр) через меню Analyzing. Термометр влево укажет на негативное влияние на цену.



Для проведения нейросетевого исследования кроме понимания его технологии необходимо иметь подходящий набор инструментов, в состав которых входят различные программы, алгоритмы, технические средства.

Программ, необходимых для анализа, чаще всего нет в наличии. Приходится их разрабатывать, искать их заменители. Одним из способов разрешения этой проблемы является комплексирование программных средств (ПС), изготовление необходимых инструментов для проведения нейросетевых исследований (НСИ).

Технология комплексирования программных средств включает в себя использование системы команд операционной системы (ОС) - так называемое «cmd-комплексирование»; расширение использования C#-программ, например, использования недокументированных возможностей алгоритмических языков - в виде запуска из C# программ операционной системы (.com, .exe, .bat, .cmd, и др.) и программ, написанных на других языках.

А для этого необходимы: соответствующая установка среды разработки, настройка командного процессора cmd и его расширение с помощью csc.

Инструменты для проведения нейросетевых исследований позволяют использовать различные нейросетевые пакеты (которые так же относятся к классу программ), использовать пространства имён языка C# для сборки инструментов, необходимых НСИ; приходится изготавливать недостающие программные инструменты (например, программы для автоматизации исследования исходных текстов программ, для соединения cs-файлов программных проектов, созданных с помощью Visual Studio; программ для автоматизации борьбы со сверхдлинными именами, образующимися при работе с Visual Studio, программные модели нейросетевых компонентов.

Решение всех этих вопросов входит в состав нейросетевого программирования, включающего в себя:

1. Нейросетевые пакеты и программные комплексы.
2. Модели нейросетевых компонентов.
3. Инструменты для проведения нейросетевых исследований
4. Технологию комплексирования программных средств.
5. Использование системы команд операционной системы
6. Использование недокументированных возможностей C#
7. Установку и настройку среды разработки, средств контроля и преобразователей

Нейросетевые пакеты и программные комплексы.

Различные организации (в том числе – и независимые группы программистов, как например Индийско-Британская группа AForge) создают и размещают в Интернет на условиях freeware пространства имён для решения задач с использованием нейронных сетей и различные модели нейросетевых компонентов.

Рассмотрим возможности двух из таких пространств имён: AForge.NET Framework, и Neuron Dot Net. Оба они написаны на алгоритмическом языке C#.

Пространство имён AForge.NET.

В Интернет его называют «от Андрея Кириллова». Расположено оно по адресу http://www.codeproject.com/KB/recipes/aforge.aspx), представляет собой удобную функциональную библиотеку, содержащую программы для нейросетевых (НС) исследований, использования генетических алгоритмов (ГА) и нечётких множеств (Fuzzy), содержит так же множество функций для обработки изображений. В библиотеке имеется так же множество примеров решения задач. Адрес сайта codeproject:

http://www.aforgenet.com

Пространство имён AForge.NET Framework (freeware), содержит два пространства имён для работы с нейросетями: AForge.Neuro Namespace и AForge.Neuro.Learning Namespace.

Пространство имен AForge.Neuro Namespace содержит интерфейсы и классы для нейронных вычислений. Это пространство имен и содержащиеся в нём подпространства содержат классы, которые позволяют как создание архитектуры популярных нейронных сетей, как и классы для обучения этих сетей. В него входят следующие классы и интерфейсы:

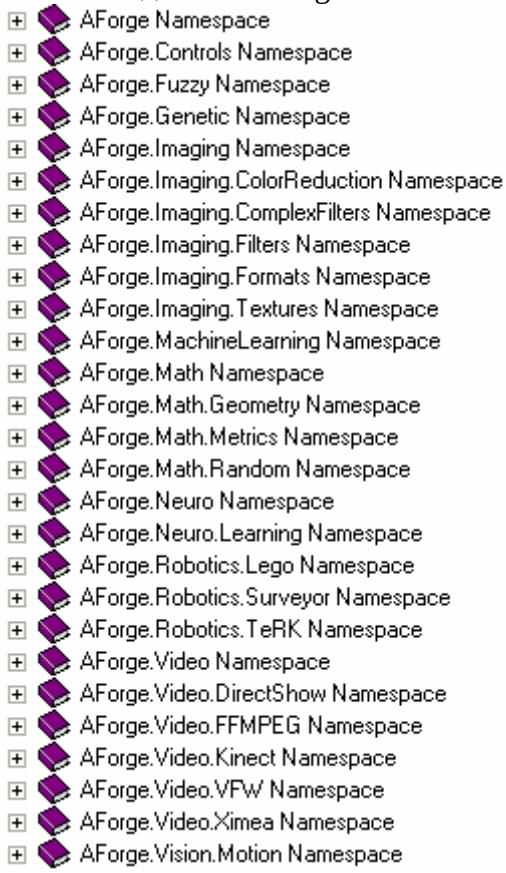
№	Class (Interfase)	Description
1	ActivationLayer	Activation layer.
2	ActivationNetwork	Activation network.
3	ActivationNeuron	Activation neuron.
4	BipolarSigmoidFunction	Bipolar sigmoid activation function.
5	DistanceLayer	Distance layer.
6	DistanceNetwork	Distance network.
7	DistanceNeuron	Distance neuron.
8	Layer	Base neural layer class.
9	Network	Base neural network class.
10	Neuron	Base neuron class.
11	SigmoidFunction	Sigmoid activation function.
12	ThresholdFunction	Threshold activation function.
13	IActivationFunction	Activation function interface.

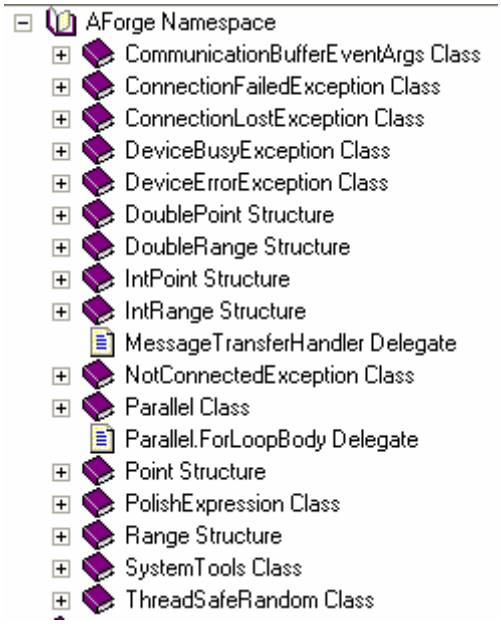
Пространство имён Forge.Neuro.Learning содержит интерфейсы и классы для обучения нейронов и нейронных сетей. В пространство имён входят как классы для обучения с учителем, так и для неуправляемого обучения:

№	Class (Interfase)	Description
1	BackPropagationLearning	Back propagation learning algorithm.
2	DeltaRuleLearning	Delta rule learning algorithm.
3	ElasticNetworkLearning	Elastic network learning algorithm.
4	EvolutionaryLearning	Neural networks' evolutionary learning algorithm, which is based on Genetic Algorithms.
5	PerceptronLearning	Perceptron learning algorithm.
6	ResilientBackpropagationLearning	Resilient Backpropagation learning algorithm.

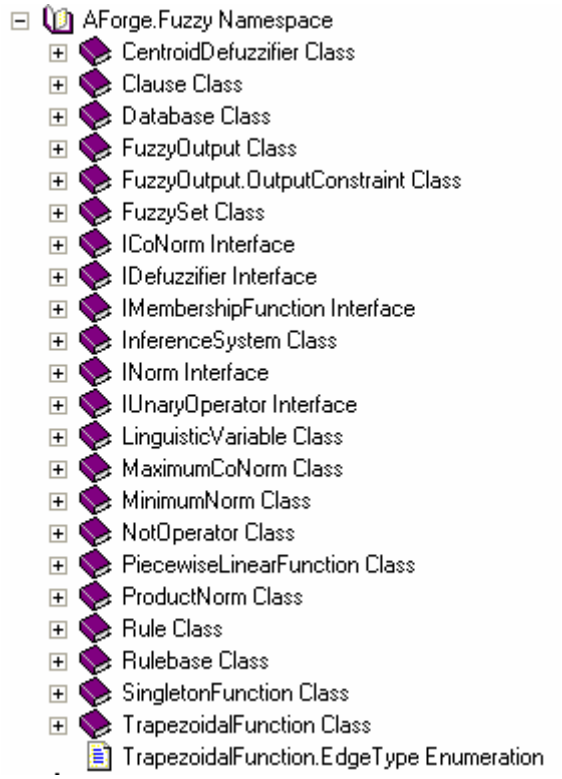
7	SOMLearning	Kohonen Self Organizing Map (SOM) learning algorithm.
8	ISupervisedLearning	Supervised learning interface.
9	IUnsupervisedLearning	Unsupervised learning interface.

Полный список компонентов пространства имён AForge.NET Framework приведен в сетевом издании «AForge.NET.chm»:

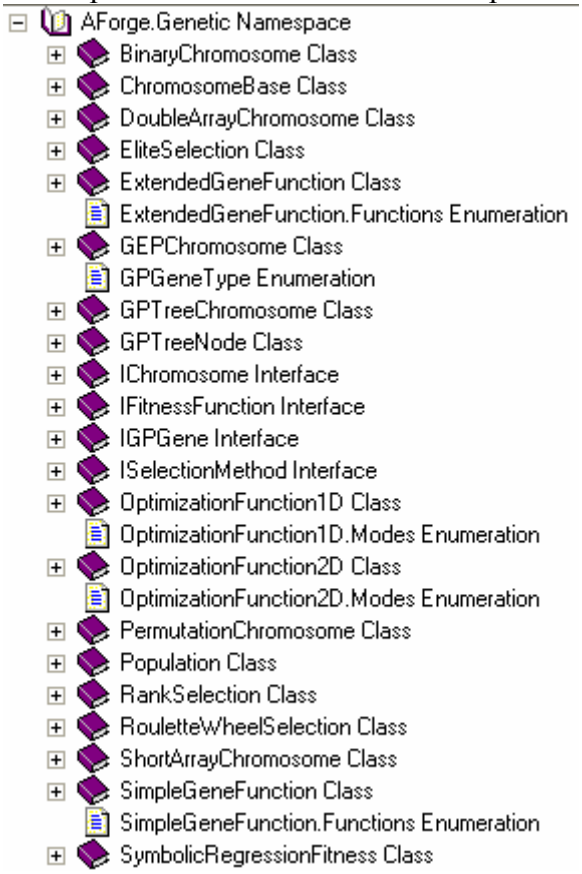




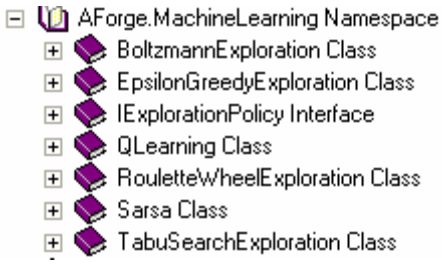
Нужно отметить, что в этот состав входят 2 очень важных самостоятельных пространства имён - для работы с нечёткими множествами (AForge.Fuzzy Namespace):



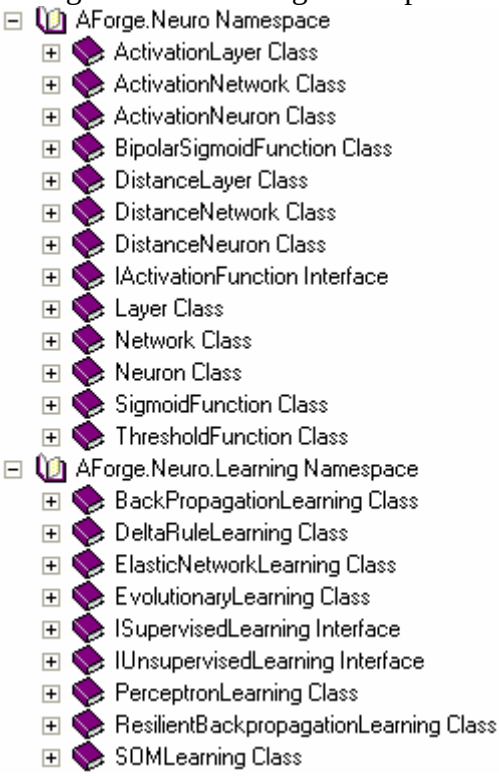
И для работы с генетическими алгоритмами:



В группе пространств имён машинного обучения (AForge.MachineLearning Namespace) находятся:



Наибольший интерес среди пространств имён AForge.NET Framework представляют приведенные ранее компоненты для работы с нейросетями: AForge.Neuro Namespace и AForge.Neuro.Learning Namespace



Среди создаваемых нейросетей есть разновидность DistanceLayer Class, которые имеют только один слой, являющийся базовым для таких сетей, как SOM, Elastic net, и так далее. Нейрон расстояния (Distance neuron) вычисляет своё выходное значение как расстояние между весовыми коэффициентами и входными значениями – в виде суммы абсолютных значений различий между значениями весов и соответствующих им значениями на входах.

Обзор пространства имён NeuronDotNet.

NeuronDotNet http://neurondotnet.freehostia.com/manual/index.html предназначен для отображения компонентов нейронной сети в библиотечные классы и интерфейсы. Подробная документация может быть загружена вместе с источником NeuronDotNet.
(Слои, разъемы, сети и принадлежность TrainingSet к интерфейсу ISerializable).

Пространство имён **NeuronDotNet.Core** включает в себя:

- **INeuron** : интерфейс для представления нейрона
- **ISynapse** : интерфейс, представляющий связи (synapse) нейронной сети
- **ILayer** : интерфейс, представляющий слои нейронной сети
- **IConnector** : интерфейс, представляющий connector (который соединяет два слоя) нейронной сети

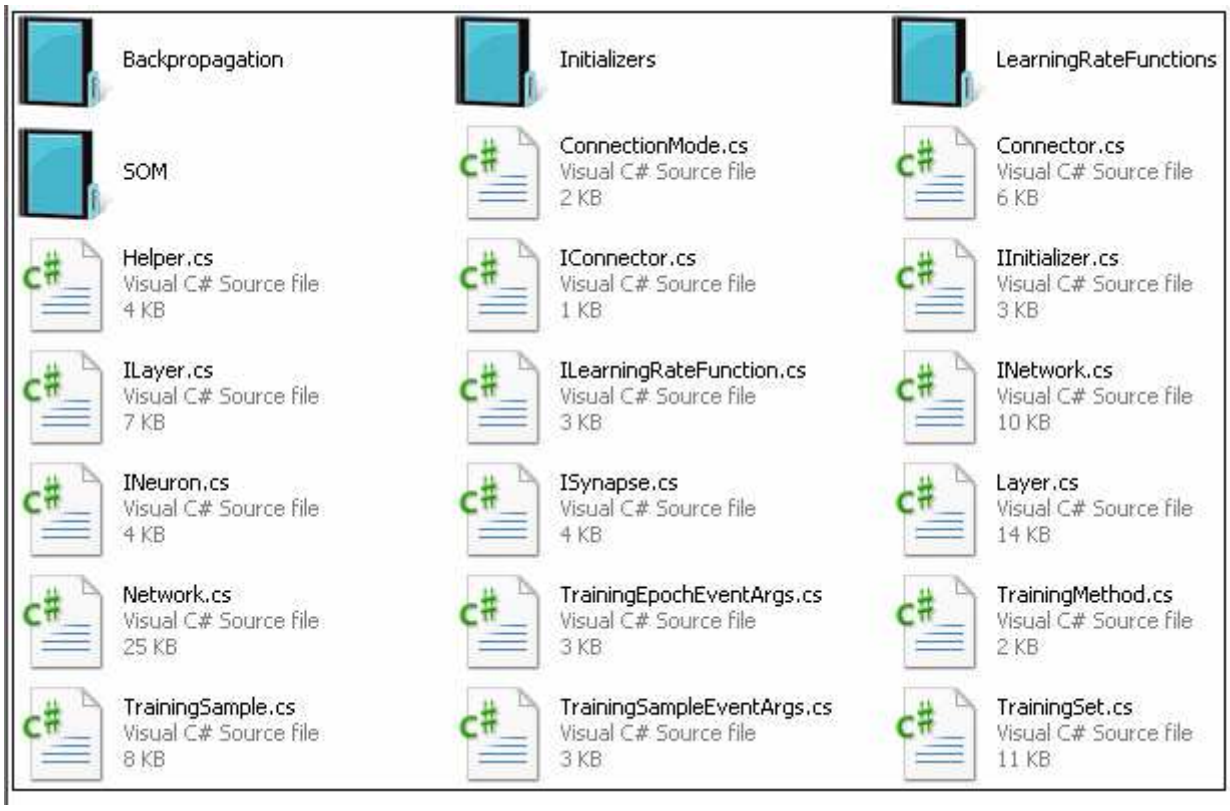
- **INetwork** : интерфейс для представления нейронной сети

- **IInitializer** : интерфейс для представления инициализатора, который определяет методы инициализации для конкретных слоёв и коннекторов.
- **ILearningRateFunction** : Интерфейс показателя функции обучаемости (Learning Rate Function interface).

- **Layer** : Абстрактная реализация 'ILayer, как генерация коллекции нейронов ('INeuron's)
- **Connector** : Абстрактная реализация 'IConnector', соединяющая два слоя
- **Network** : Абстрактный базовый класс основной нейронной сети. Он реализует 'INetwork'.
- **TrainingSample** : Обучающая выборка, представляющая пару векторов: вход – и ожидаемый выход.
- **TrainingSet** :Тренировочная выборка, как коллекция пренировочных примеров..

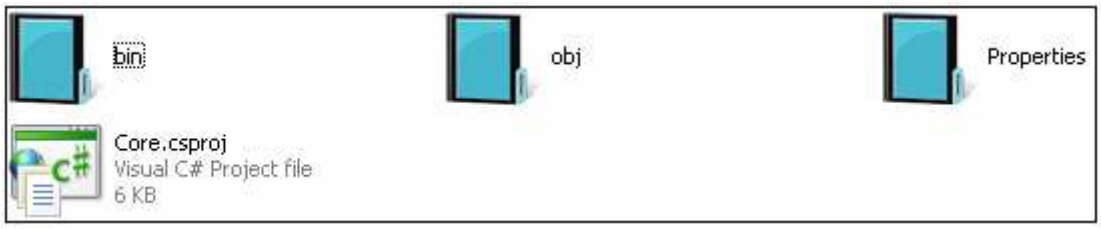
- **TrainingMethod** : Указание, какой метод обучения используется: с учителем, или без учителя (supervised or unsupervised).
- **ConnectionMode** : Указание на то, какой коннектор используется: один-один, или полный.

Весь пакет NeuronDotNet в виде исходных текстов собран в папке Core, содержащей:



Здесь видно, что пространство имён Core содержит 17 cs-файлов и 4 пространства имён в виде папок: Backpropagation, Initializers, LearningRateFunctions, SOM, которые представляют собой ещё 4 пространства имён (их имена совпадают с именами папок).

Кроме того, в дистрибутиве пространства имён Core содержится программный проект:



В его состав входит файл NeuronDotNet.Core.dll – откомпилированная динамическая библиотека пакета NeuronDotNet . Для работы с ней служат стандартные кодовые фрагменты.

Некоторые полезные кодовые фрагменты.

Данный ниже код создает сеть backpropagation с линейным входным слоем, содержащем десять нейронов, сигмоидный промежуточный слой, содержащий пять нейронов и сигмоидный выходной слой, содержащий семь нейронов.

```
LinearLayer inputLayer = new LinearLayer(10);
SigmoidLayer hiddenLayer = new SigmoidLayer(5);
SigmoidLayer outputLayer = new SigmoidLayer(7);
new BackpropagationConnector(inputLayer, hiddenLayer);
new BackpropagationConnector(hiddenLayer, outputLayer);
BackpropagationNetwork network = new BackpropagationNetwork(inputLayer, outputLayer);
```

Для того, чтобы создать сеть, сначала создаются слои (layers), затем соединяются слои, создавая разъемы (connectors), затем создают сеть (net), определяя желаемый входной (inputlayer) и выходной (outputlauer) слои.

Пользователь НЕ должен модифицировать структуру сети после её создания.

Нужно помнить, что входной слой сети обратного распространения всегда линейный, чтобы не модифицировать пользовательский вход при добавлении bias или activation функций.

Коды для создания сети Kohonen SOMмогут быть следующими:

```
KohonenLayer inputLayer = new KohonenLayer(2);
KohonenLayer outputLayer = new KohonenLayer(new Size(neuronCount, 1));
KohonenConnector connector = new KohonenConnector(inputLayer, outputLayer);
KohonenNetwork network = new KohonenNetwork(inputLayer, outputLayer);
```

При необходимости свойства некоторых слоёв или connector могут быть изменены.

```
outputLayer.Initializer = new ZeroFunction();
outputLayer.NeighborhoodFunction = new GaussianFunction(10);
outputLayer.IsRowCircular = true;

connector.Initializer = new RandomFunction(-1, 1);
```

Обучающая функция связана с каждым слоем в сети. Мы можем модифицировать свойства индивидуально для каждого слоя. Если мы хотим использовать единственную функцию для всех слоев, мы можем использовать метод SetLearningRate сети.

```
network.SetLearningRate(0.3);
```

Для обучения сети необходимы обучающие примеры. Коллекция обучающих примеров объединяется в обучающую выборку (файл), который и используется для обучения сети.

Сеть обучается на всех обучающих примерах один-за-одним, повторяя их много раз, пока сеть не обучится полностью.

```
TrainingSet trainingSet = new TrainingSet(2, 1);
trainingSet.Add(new TrainingSample(new double[2] { 0d, 0d }, new double[1] { 0d }));
trainingSet.Add(new TrainingSample(new double[2] { 0d, 1d }, new double[1] { 1d }));
trainingSet.Add(new TrainingSample(new double[2] { 1d, 0d }, new double[1] { 1d }));
trainingSet.Add(new TrainingSample(new double[2] { 1d, 1d }, new double[1] { 0d }));
network.Learn(trainingSet, 1000);
```

Метод Initialize ре-инициализирует (unlearns) связи и нейроны, подготавливая их к обновлению обучения. BeginEpochEvent, EndEpochEvent, BeginSampleEvent и EndSampleEvent - некоторые события, которые помогают потребителю, чтобы изучать поведение сети во время обучения.

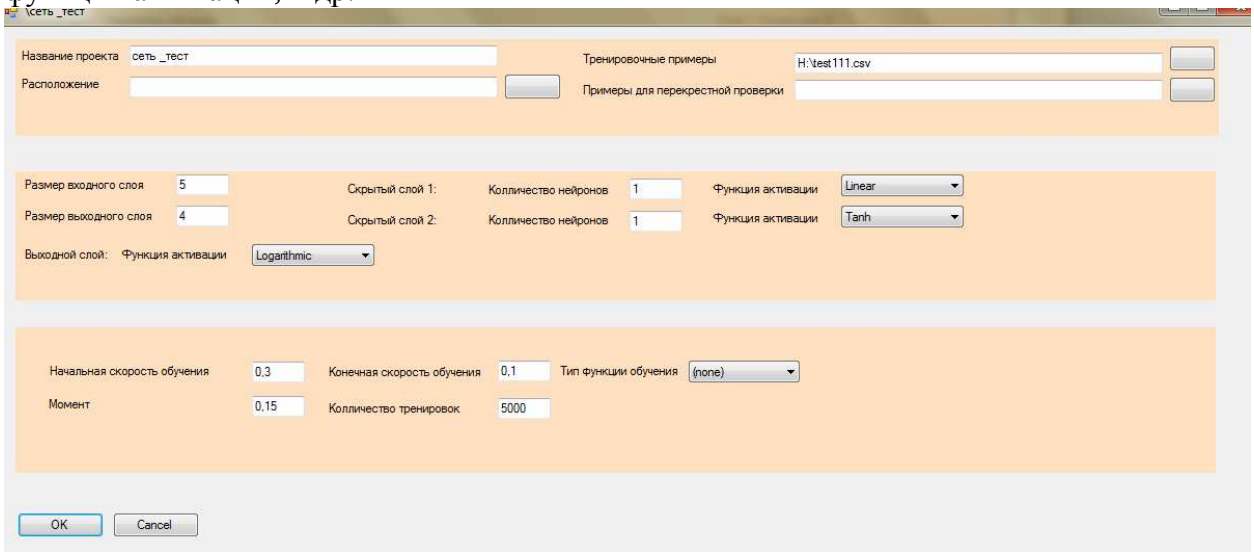
Обучающая выборка и все сети используют интерфейс 'ISerializable', благодаря чему могут быть сохранены в файле для последующего использования

Структура папок пакета Neuron Dot Net:
Серийный номер тома: 0006EFC4 3009:ADA7
C:.

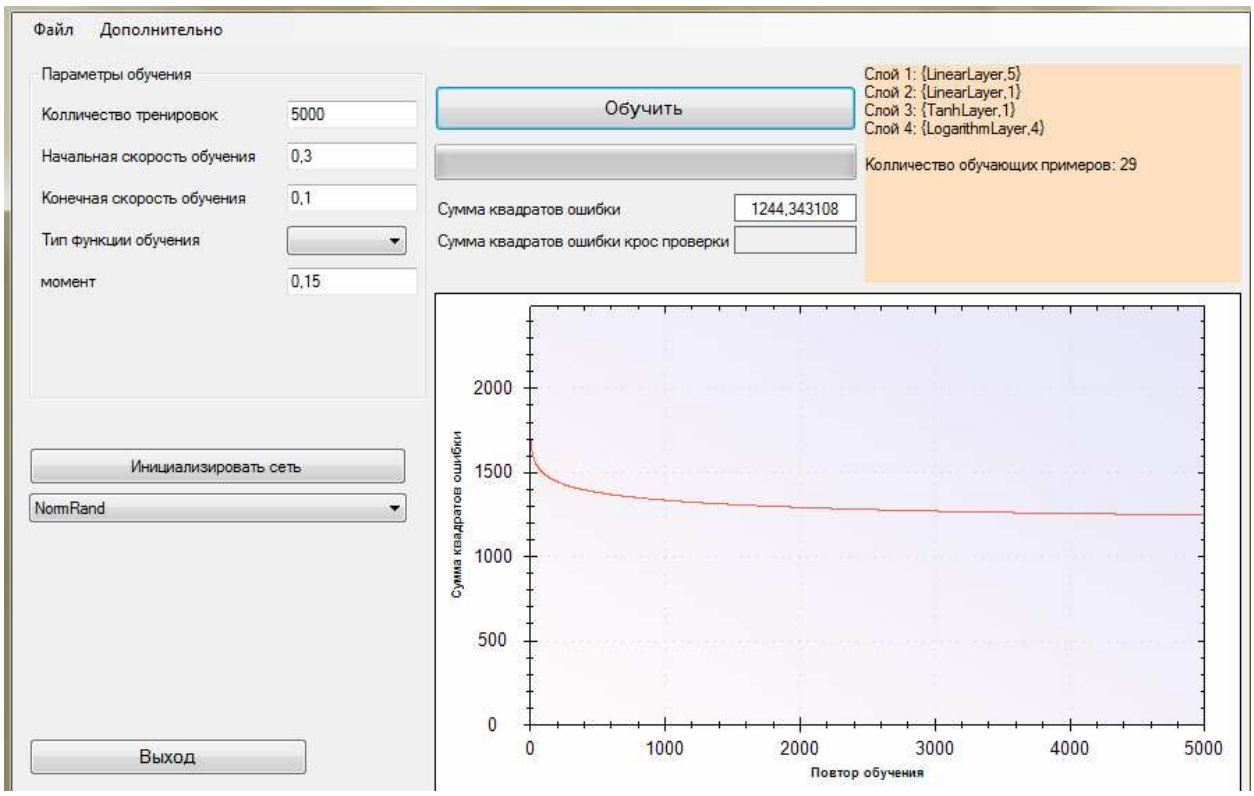
```
    ConnectionMode.cs
    Connector.cs
    Core.csproj
    Helper.cs
    IConnector.cs
    IInitializer.cs
    ILayer.cs
    ILearningRateFunction.cs
    INetwork.cs
    INeuron.cs
    ISynapse.cs
    Layer.cs
    Network.cs
    TrainingEpochEventArgs.cs
    TrainingMethod.cs
    TrainingSample.cs
    TrainingSampleEventArgs.cs
    TrainingSet.cs

├───Backpropagation
        ActivationLayer.cs
        ActivationNeuron.cs
        BackpropagationConnector.cs
        BackpropagationNetwork.cs
        BackpropagationSynapse.cs
        LinearLayer.cs
        LogarithmLayer.cs
        SigmoidLayer.cs
        SineLayer.cs
        TanhLayer.cs

├───bin
```

```
    └──Release
        NeuronDotNet.Core.dll
        NeuronDotNet.Core.xml

├──Initializers
    ConstantFunction.cs
    NguyenWidrowFunction.cs
    NormalizedRandomFunction.cs
    RandomFunction.cs
    ZeroFunction.cs

├──LearningRateFunctions
    AbstractFunction.cs
    ExponentialFunction.cs
    HyperbolicFunction.cs
    LinearFunction.cs

├──Properties
    AssemblyInfo.cs

└──SOM
    INeighborhoodFunction.cs
    KohonenConnector.cs
    KohonenLayer.cs
    KohonenNetwork.cs
    KohonenSynapse.cs
    LatticeTopology.cs
    PositionNeuron.cs

    └──NeighborhoodFunctions
        GaussianFunction.cs
        MexicanHatFunction.cs
```

Для работы с такой структурой папок создаётся пользовательский интерфейс, например содержащий две формы: первая – для отображения названия программного проекта и его размещения; указания адреса тренировочных примеров; количества и размеров слоёв, функций активации, и др.



И вторая – для иллюстрации хода обучения



Пример создания перцептрона для решения задачи AND.

Создадим перцептрон с одним скрытым слоем с 2 нейронами, 2 входными и 1 выходным нейронами. Функции активации нейронов будут сигмоидальными.
В качестве функции обучения выберем экспоненциальную, метод обучения BackPropagation.
Предполегаем, что 100 эпох должно быть достаточно для качественного обучения.
Обучающая выборка – это всевозможные пары из 0 и 1, то есть 4 примера:

0 AND 0
1 AND 0
0 AND 1
1 AND 1

Код программы с комментариями.

```
using System;
using NeuronDotNet.Core;
using NeuronDotNet.Core.Backpropagation;
using NeuronDotNet.Core.LearningRateFunctions;
namespace Network
{
    class Program
    {
        static void Main(string[] args)
        {
            LinearLayer inputLayer = new LinearLayer(2); //создаем вх. линейный слой с 2 нейронами
            SigmoidLayer hiddenLayer = new SigmoidLayer(2); //создаем скрытый слой с 2 нейронами
            SigmoidLayer outputLayer = new SigmoidLayer(1); //создаем выходной слой с 1 нейроном
            new BackpropagationConnector(inputLayer, hiddenLayer); //делаем
```

```
связи между 1 и 2 слоем
            new BackpropagationConnector(hiddenLayer, outputLayer); //связи
между скрытым и выходом
            var andNetwork = new BackpropagationNetwork(inputLayer,
outputLayer); //создаем саму нейронную сеть с обратным распространением
ошибки и указываем входные и выходные параметры
            andNetwork.SetLearningRate(new ExponentialFunction(1, 0.01));
//задаем функцию обучения
            TrainingSet trainingSet = new TrainingSet(2, 1); //создаем
обучающую выборку для AND
            trainingSet.Add(new TrainingSample(new double[] { 0d, 0d }), new
double[] { 0d }));
            trainingSet.Add(new TrainingSample(new double[] { 0d, 1d }), new
double[] { 0d }));
            trainingSet.Add(new TrainingSample(new double[] { 1d, 0d }), new
double[] { 0d }));
            trainingSet.Add(new TrainingSample(new double[] { 1d, 1d }), new
double[] { 1d }));
            andNetwork.EndEpochEvent += delegate (object net,
TrainingEpochEventArgs ev)
{
    if ((ev.TrainingIteration + 1) % 50 == 0)
        Console.WriteLine("epoch {0}: error = {1}", ev.TrainingIteration + 1,
                          andNetwork.MeanSquaredError);
};
//выводим ошибку сети на каждой 50-ой эпохе обучения (здесь
среднеквадратичная MSE)
            andNetwork.Learn(trainingSet, 100); //задаем число эпох
            Console.WriteLine("Таблица истинности для операции И:");
            for (int i = 0; i < 4; i++)
                Console.WriteLine("{0} AND {1} = {2}", i & 1, i >> 1,
                    andNetwork.Run(new double[] { i & 1, i >> 1 })[0]);
//вывод результата обучения
        }
    }
}
```

Для исполнения программы 1.cs её нужно откомпилировать с указанием имени используемой библиотеки NeuronDotNet.Core.dll.

Инструкция компилятору:

```
>>H:\1>csc  /reference:NeuronDotNet.Core.dll 1.cs
Microsoft (R) Visual C# 2010 Compiler version 4.0.30319.1
Copyright (C) Microsoft Corporation. All rights reserved.
//Запуск откомпилированной программы
>>1.exe
epoch 50: error = 0,165278577780727
epoch 100: error = 0,000368177622090322
Таблица истинности для операции И:
0 AND 0 = 0,0112435704326343
1 AND 0 = 0,0191074657022637
0 AND 1 = 0,0182365157133779
1 AND 1 = 0,977562543494591
H:\1>
```

Использование нейросетевой компонентной модели Sharky Neural Network в качестве основы лабораторного изучения многослойности перцептрона.

Среди нейросетевых пакетов freeware в Интернет можно найти группу

демонстрационных программ, использование которых позволяет наглядно демонстрировать студентам различные свойства нейросетей. Есть среди этой группы простые, позволяющие демонстрировать отдельные свойства нейросетей, такие, как Sharky, Kohonen, EasyDemo, и сложные пакеты – для демонстрации групп свойств, к которым можно отнести и такой коммерческий пакет, как Матлаб.

Sharky Neural Network – это компьютерная программа фирмы SharkTime Software (http://www.sharktime.com) для игровой демонстрации возможностей нейросетевого классификатора. Программа freeware, работает под ОС Windows разных версий.

Программа реализует нейронную сеть типа многослойного перцептрона, предназначенную для классификации 2D-точек в два различных класса, жёлтый и синий. Каждое множество 2D-точек представляет собой геометрическую фигуру (форму) - круг, квадрат, бриллиант, волну, луну или другую фигуру. Программа при классификации не определяет форму. Она просто делит все точки на две группы: синие и жёлтые. Геометрическая форма распознаваемых фигур при этом проявляется при визуализации результата классификации.

Исходные данные можно загрузить только в виде заказанного образа из нескольких имеющихся заготовок (xor, circle, square, diamond, ring, moon, wave, и др.). На сайте подготовлены для загрузки дополнительные файлы «AI.points», «cn.points», «N.points», «Two_Spirals_Cartesian.points» и «Two_Spirals_Radial.points».

Программа позволяет вносить изменения в исходные данные: добавлять, удалять, загружать или сохранять точки. Комбинация клавиш *Ctrl + Left Click* позволяет работать в режиме spray.

При активизации программы на экране появляется основное окно (рис. 1).

Основное меню программы содержит 5 пунктов, содержащихся в первой строке основного экрана: Network - нейросеть, Shape - тип (форма) исходных данных, Points - работа с исходными точечными множествами, View - способ демонстрации данных на экране, Help - помощь.

С нейросетью можно производить следующие действия: «Запуск», «Остановка» (Stop), «Обновление сети» (Reset Network), «Обучение» (Learn), «Выход из программы» (Exit). Команды для Запуска и Остановки вынесены на вторую строку экрана (Learn и Stop), причём, кнопка запуска содержит название операции, которая будет выполняться (в данном примере выполняться будет обучение (Learn)) - какая именно операция будет выполняться, выбирается в пункте меню Network.

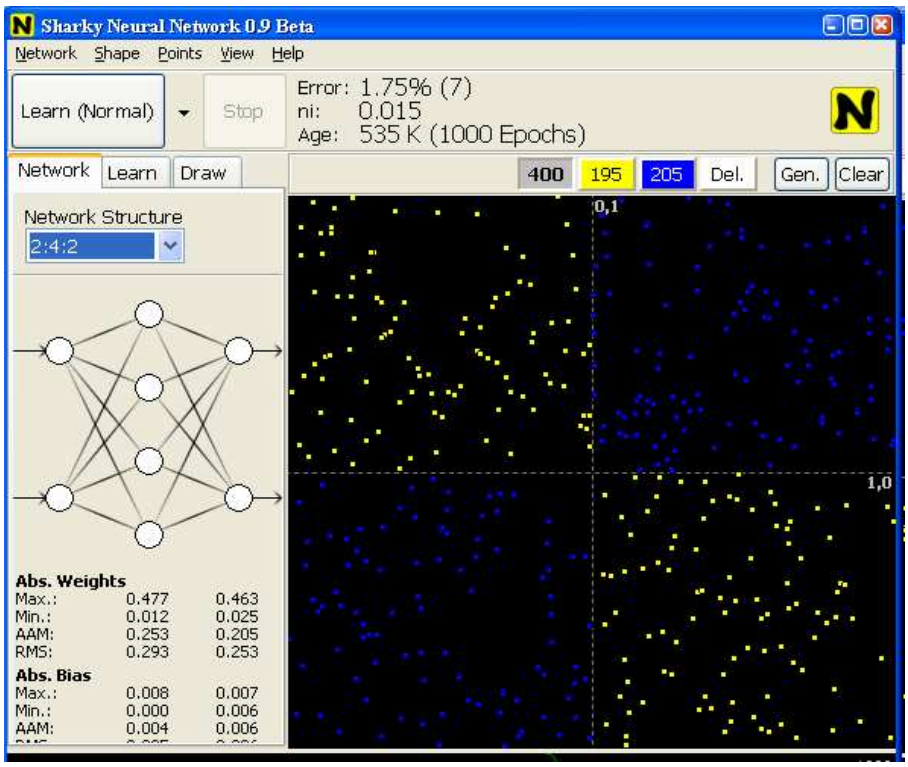


Рис. 1.

Структура нейросети задаётся в виде количества слоёв перцептрона и количества нейронов в каждом слое (входной и выходной слои всегда содержат по 2 нейрона).

В программе используется структура нейросети 2:...:2**.** Первая цифра 2 означает «два входа», так как каждая 2D-точка имеет два значения – x и y. Символ 2 в конце означает «2 выхода», так как эта сеть классифицирует на 2 различных класса (жёлтый, синий).

Для задания типа сети нужную структуру (Network Structure) предлагается найти и отметить в таблице (3 строка экрана). Тип сети выбирается из следующего списка:

2:2	2:10:2	2:2:2:2	2:5:4:2	
2:1:2	2:11:2	2:3:3:2	2:3:10:2	
2:2:2	2:12:2	2:4:4:2	2:10:3:2	
2:3:2	2:13:2	2:5:5:2	2:4:10:2	2:4:3:4:2
2:4:2	2:14:2	2:10:10:2	2:10:4:2	2:4:4:4:2
2:5:2	2:15:2	2:3:4:2	2:5:10:2	2:3:4:5:2
2:6:2	2:20:2	2:4:3:2	2:10:5:2	2:5:4:3:2
2:7:2	2:25:2	2:3:5:2	2:2:2:2:2	2:4:5:4:2
2:8:2	2:50:2	2:5:3:2	2:3:3:3:2	2:5:4:5:2
2:9:2	2:100:2	2:4:5:2	2:3:4:3:2	2:5:5:5:2

где 2:2 означает двуслойный перцептрон с 2 нейронами во входном слое и 2 – в выходном. 2:9:2 означает трёхслойный перцептрон с 9 нейронами в промежуточном слое; а 2:5:5:5:2 означает пятислойный перцептрон с 5 нейронами в каждом из 3 промежуточных слоёв.

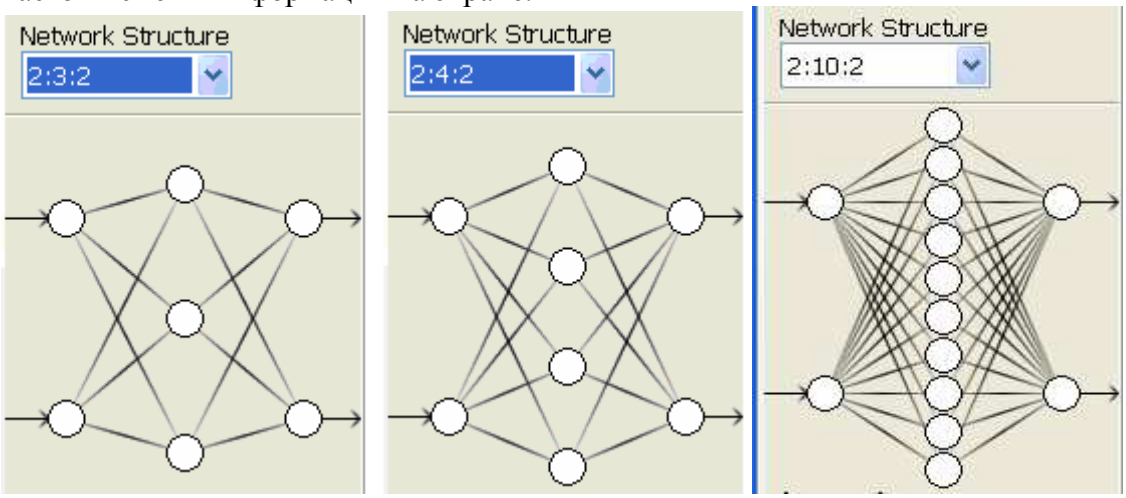
Каждый нейрон имеет смещение (*bias*) и использует двуполярный сигмоид в качестве функции активации ($f(x)=2/(1+e^{-\beta x})-1$).

Выбранная архитектура сети наглядно отображается схемой в левой части экрана (Рис. 2).

Режим обучения настраивается при нажатии кнопки Learn в 3 строке экрана. Для обучения существенными являются такие параметры, как порядок предъявления образцов (Order), фиксация ошибок (Premphase Error вкл/выкл), проверка (Verify), числовые значения ni (min и max), auto decrease (фиксированное значение 0,912), down rate, momentum. Кроме того, устанавливаются такие параметры, как режим обучения (Hard, Normal или Soft) и количество эпох (от 1 до 10000).

Результат представляется в виде рисунка 2D Graph (пункт меню View), на котором могут быть отображены: оси координат (Coordinate Axis), только множества точек (Points), точки и ответ нейросети (Points and Network Answer), и т.д. Для облегчения восприятия различные виды информации окрашиваются в разные цвета. Предусмотрена регулировка

частоты смены информации на экране.



и т.д.

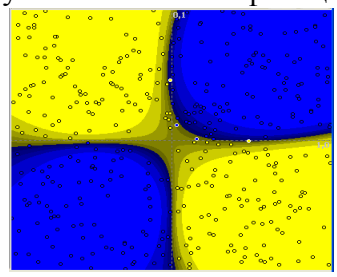
Рис. 2.

При решении задачи XOR для очистки экрана в меню Points выбирается пункт Clear Points. Затем Shape -> XOR. При выборе XOR зона графики заполняется множествами точек, синих – 1 и жёлтых – 2 (рис. 1).

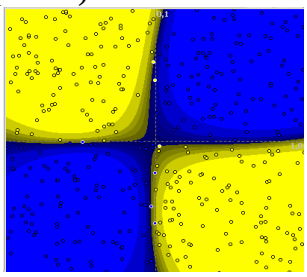
Координаты точек (их всего 400) можно сохранить в файле (Points -> Save Points As…):

-0.887500 -0.046000 2
-0.772500 -0.284750 2
-0.796000 0.637500 1
-0.054500 0.662750 1
…

После нажатия кнопки Learn через некоторое время (может потребоваться несколько десятков сек.) получим результат в виде жёлтой и синей зон эрана (рис. А). Незначительные изменения настроек в рассматриваемом примере позволяют изменить результат в сторону уточнения классификации (рис. В):



(Рис. А)

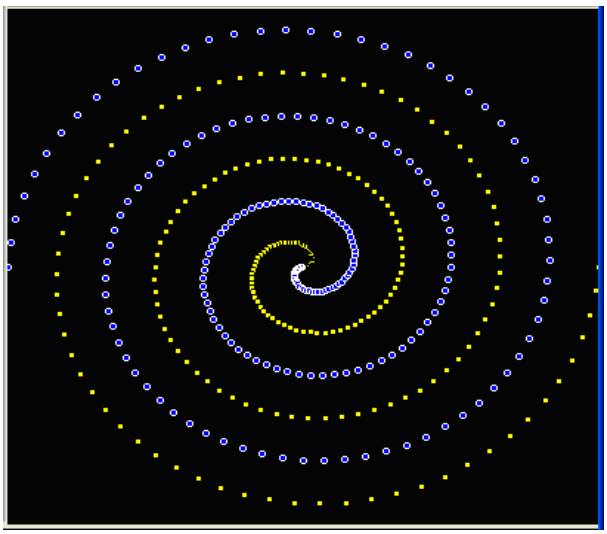


(Рис. В)

Изменяя структуру нейросети (количество слоёв и количество нейронов в каждом слое), количество эпох, параметры обучения (критерии окончания), способы вывода информации на экран и корректируя предложенные графические образы, можно увидеть, как влияют различные параметры на распознавание исследуемой совокупности точек.

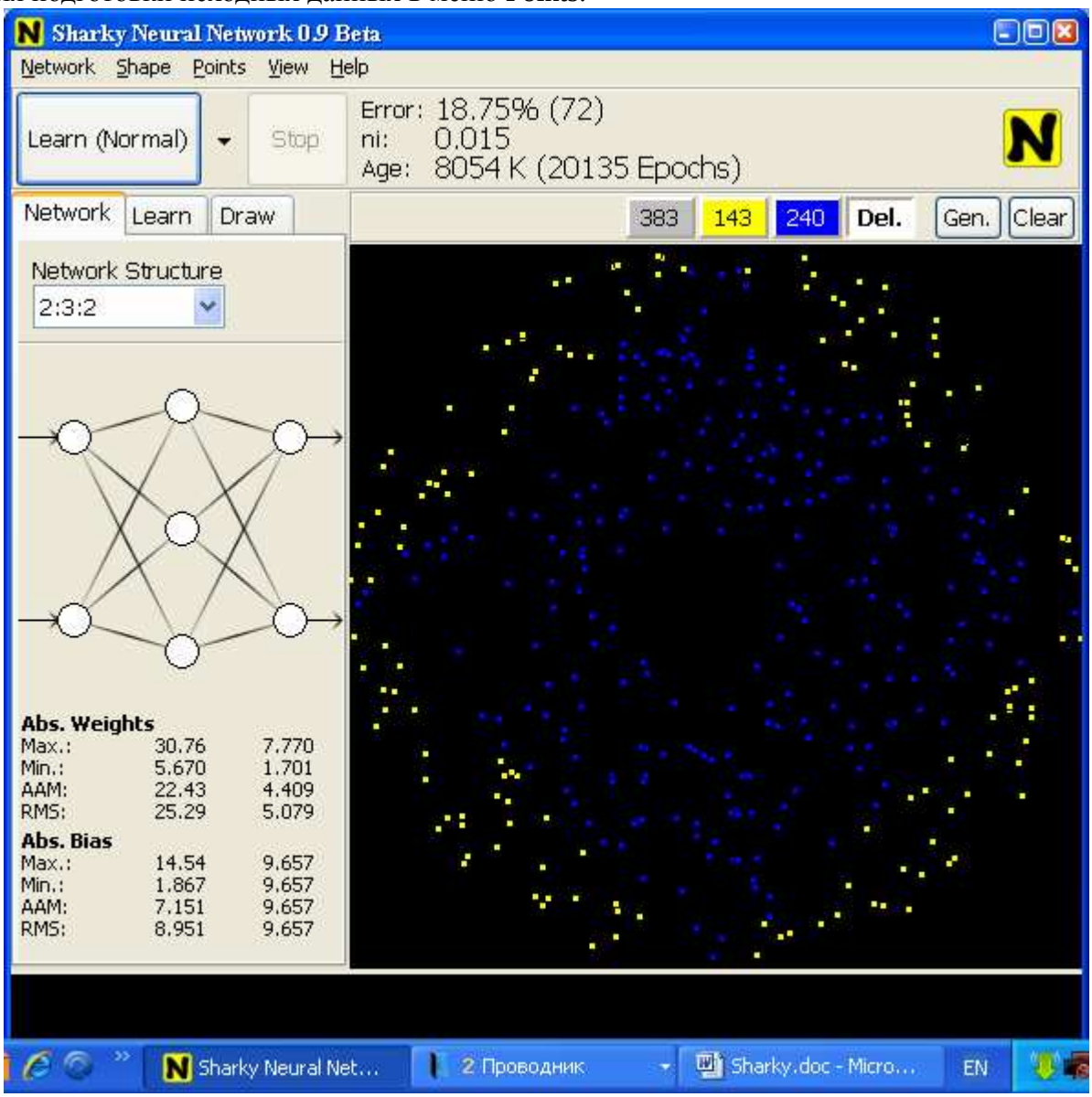
При наличии обучающей и тестовой совокупностей точек по графикам в нижней части основного экрана можно увидеть момент переобучения сети и скорректировать необходимое количество эпох.

На сайте SharkTime Software (http://www.sharktime.com) есть другие примеры, на основе которых можно проверить возможности многослойного перцептрона при классификации различных фигур (образов), например, типа двойной спирали:

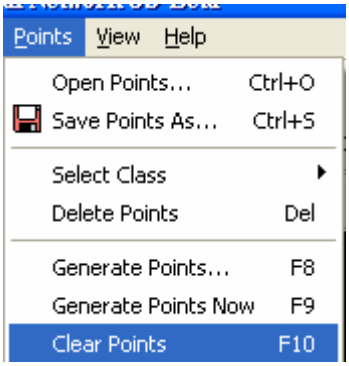


Подготовка исходных данных.

Для подготовки исходных данных в меню Points:

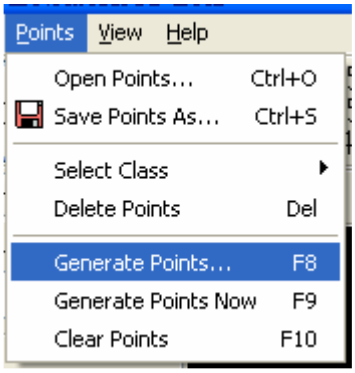


В Points выбирается пункт Clear Points.

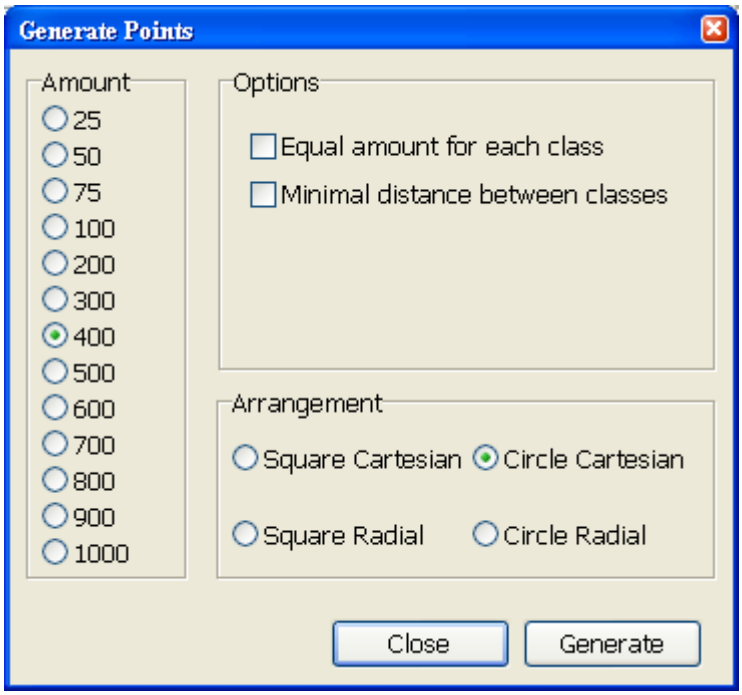


Получаем пустое окно.

В Points выбираем Generate Points

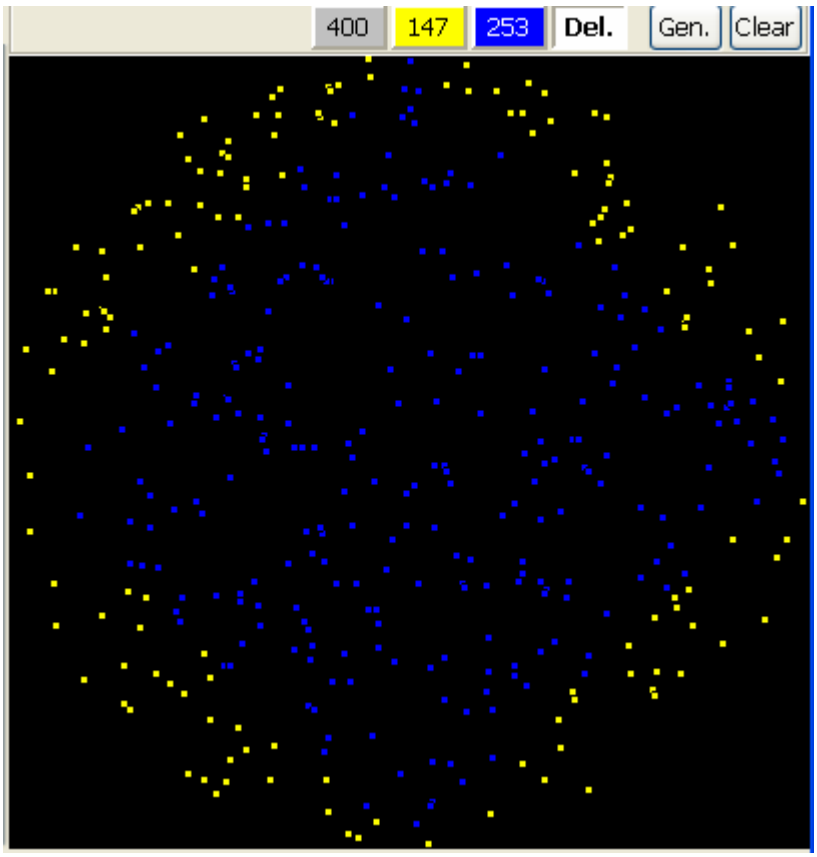


Появляется

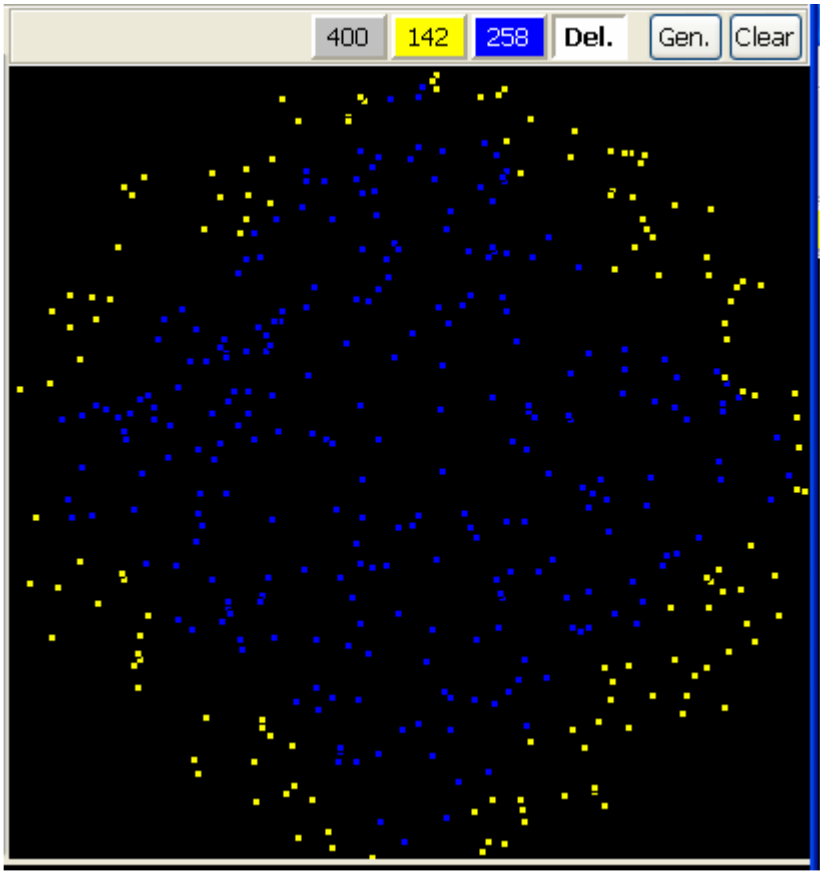


1

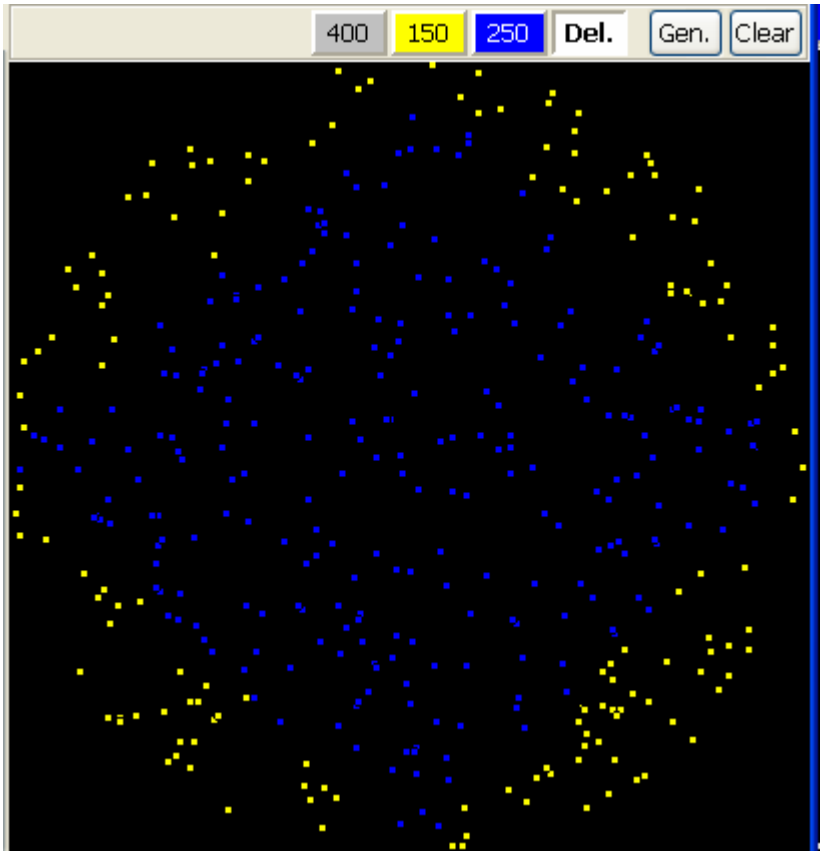
И после настройки нажимая Generate, генерируем новые группы точек:



2

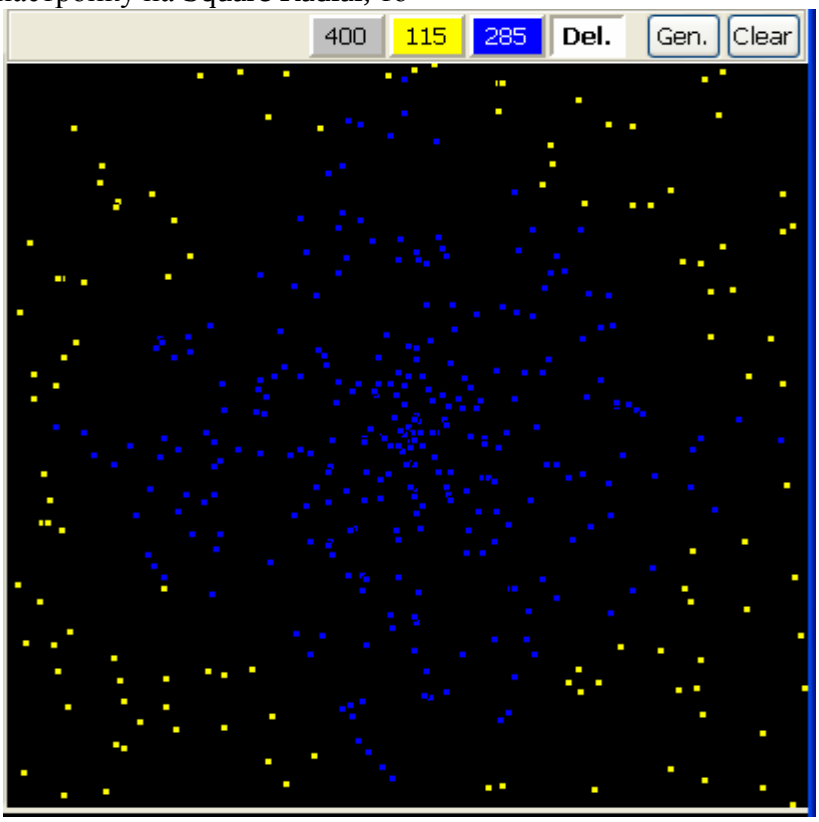


3



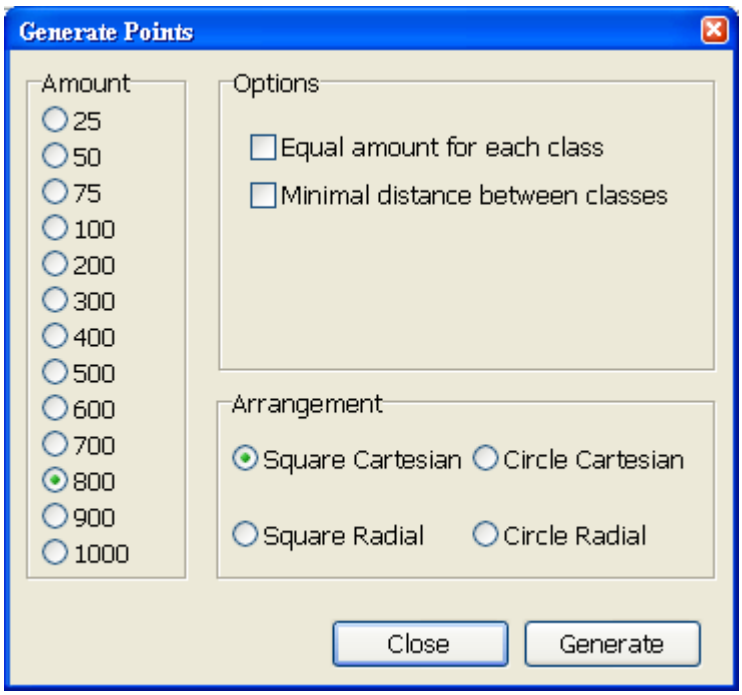
4

Если изменить настройку на Square Radial, то

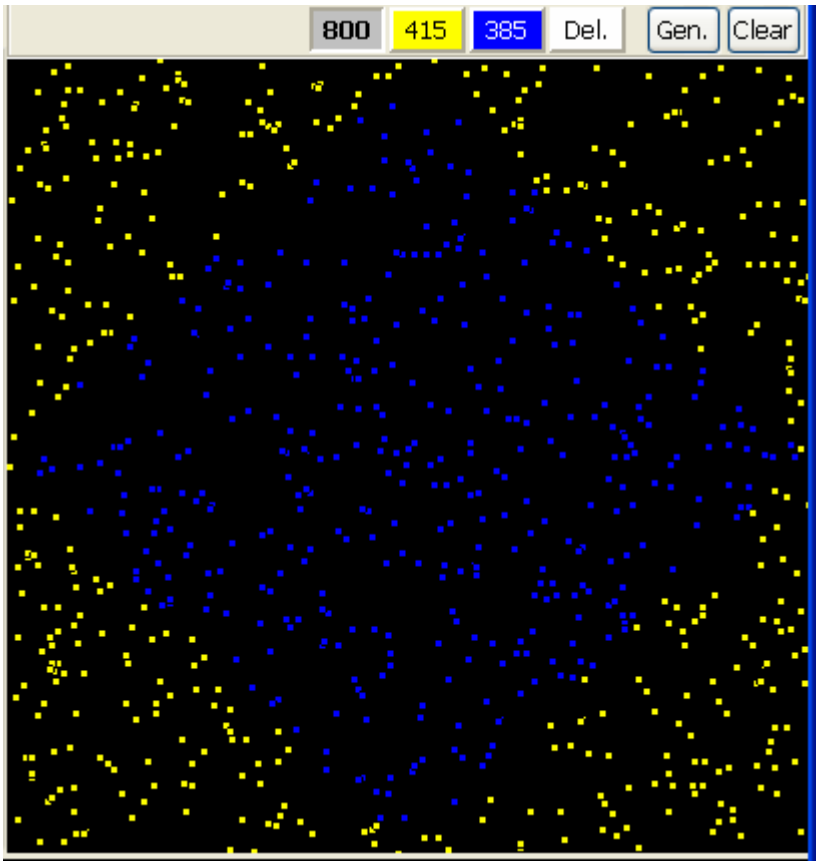


5

Или после изменения

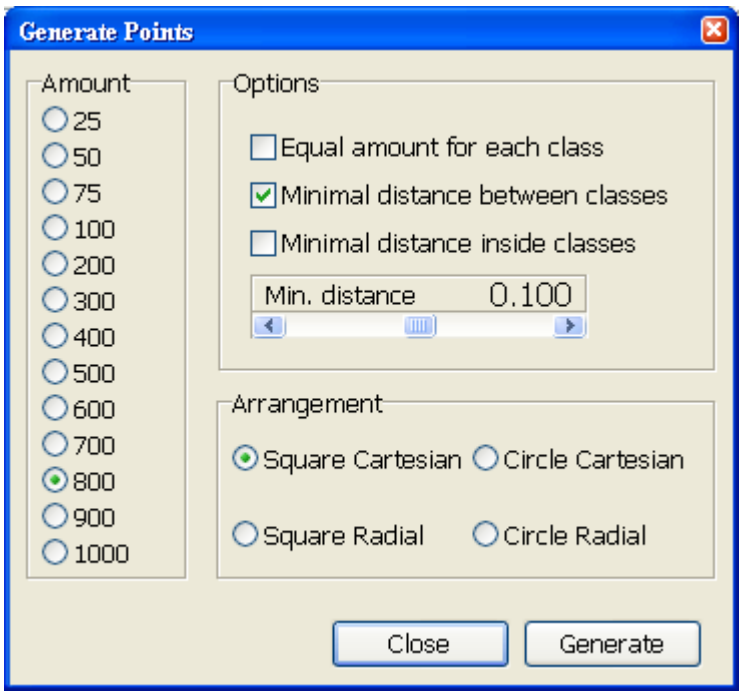


6

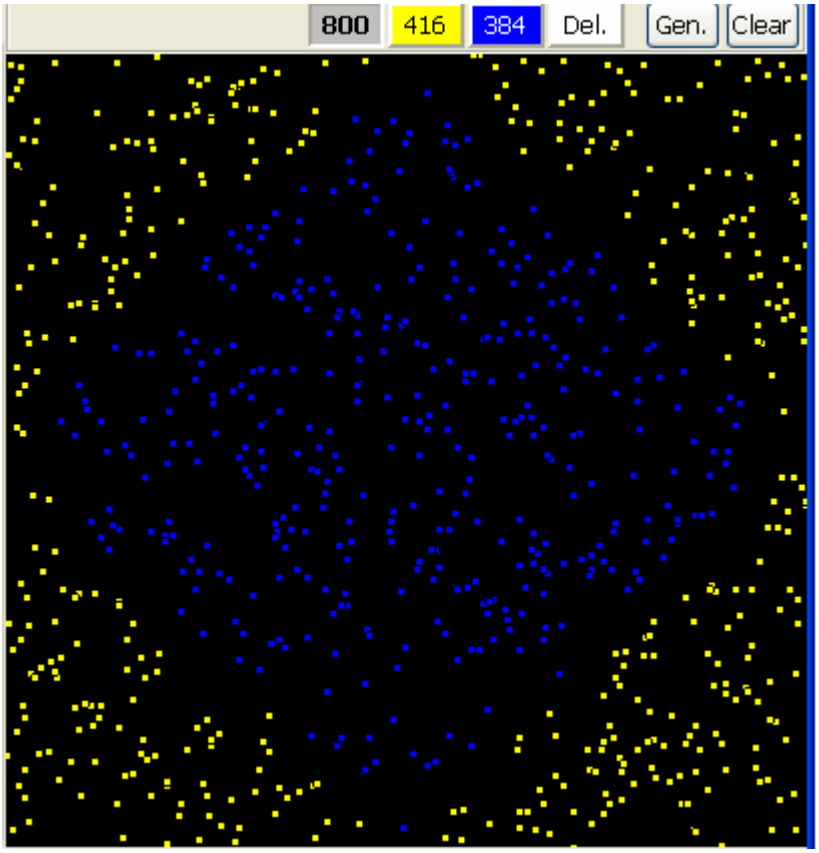


7

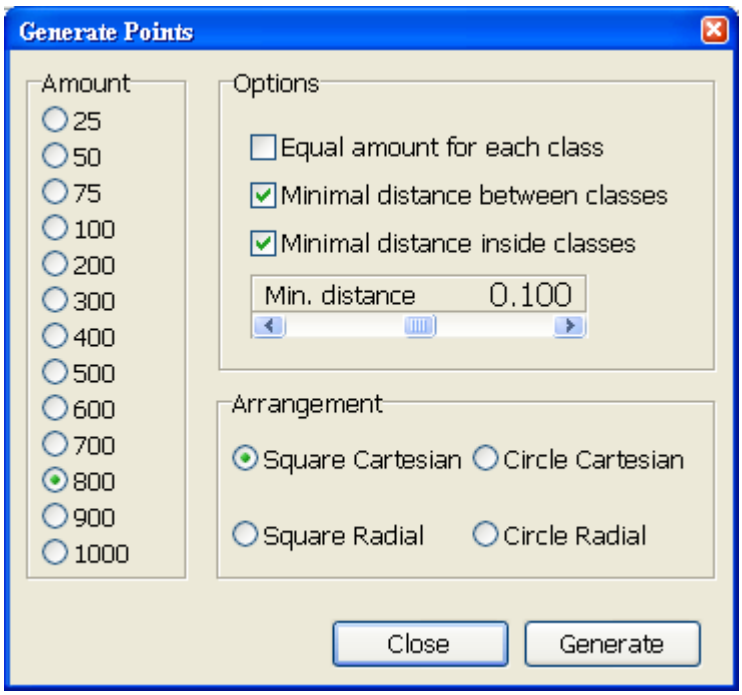
А после добавления условия:


Generate Points
Amount
25
50
75
100
200
300
400
500
600
700
800
900
1000
Options
Equal amount for each class
Minimal distance between classes
Minimal distance inside classes
Min. distance
0.100
Arrangement
Square Cartesian
Circle Cartesian
Square Radial
Circle Radial
Close
Generate

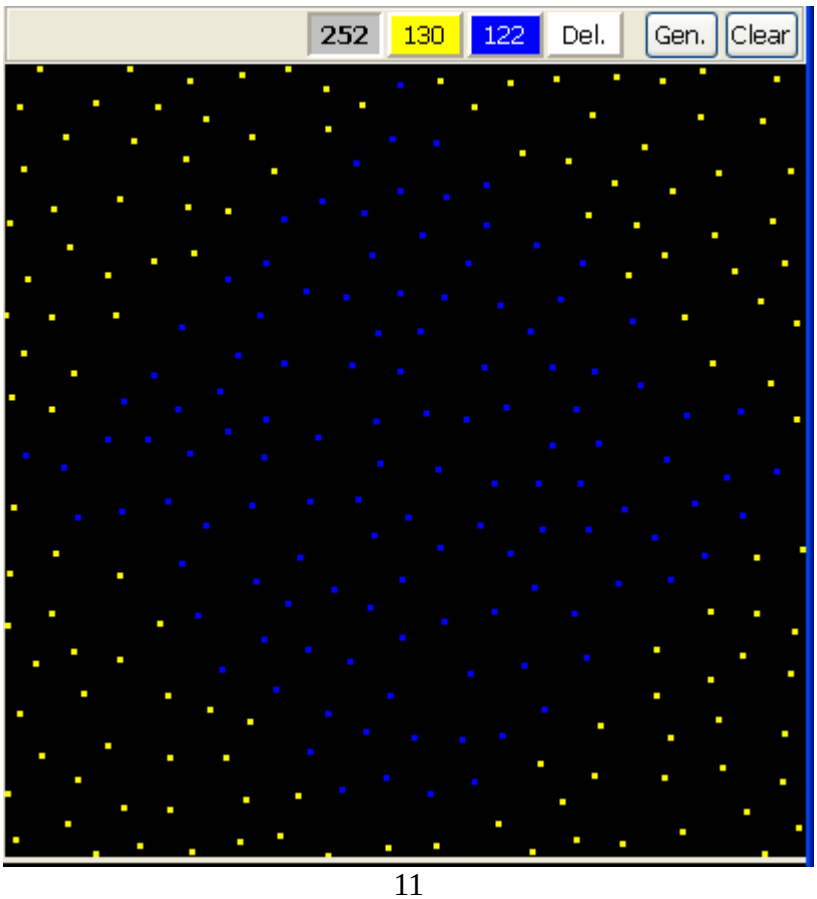
8


800
416
384
Del.
Gen.
Clear

9



10



11

Использование программы Sharky для контрастирования нейронной сети.

Контрастирование нейронной сети служит для упрощения её архитектуры, уменьшения числа входных сигналов и снижения требований к точности и количеству входных сигналов.

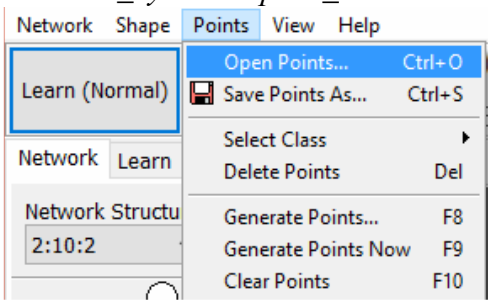
Последовательность работы:

1. Создать, обучить и проверить нейросеть в Sharky, решающую задачу классификации точек “two spirals Cartesian”. Привести и расшифровать графики, характеризующие обучение.
2. Изменяя различные параметры сети, записывать характеристики, получающиеся во всех опытах, и на их основе выявить минимальное число слоев и нейронов в них, когда задача еще решается.
3. За счет изменения количества эпох определить оптимальное их значение для обучения рассмотренного типа сети.
4. Описать, как влияют регулируемые параметры на эффективность обучения.
5. Выявить условия, при которых происходит переобучение сети.

Выполнение.

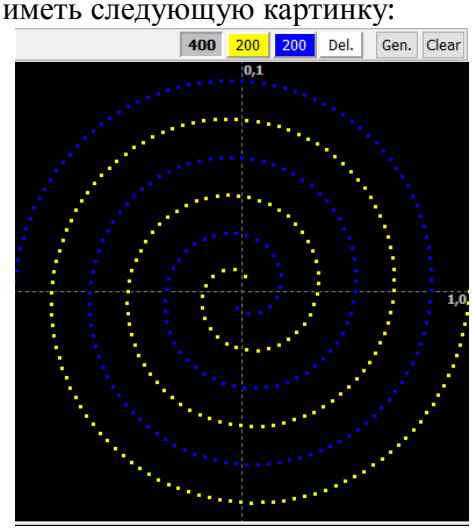
1.Создание и обучение нейросети.

Для начала загрузим в Sharky свой набор точек. Для этого в верхнем меню выбираем пункт: *Points→Open Points→_нужный файл_*

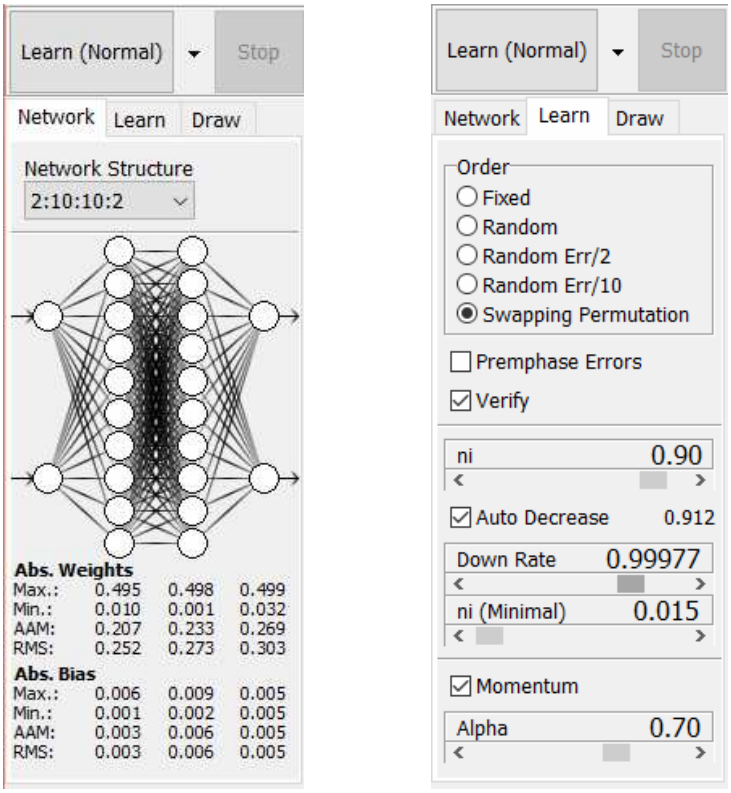


Выбор множества «Two Spirals Cartesian»

После выбора мы будем иметь следующую картинку:

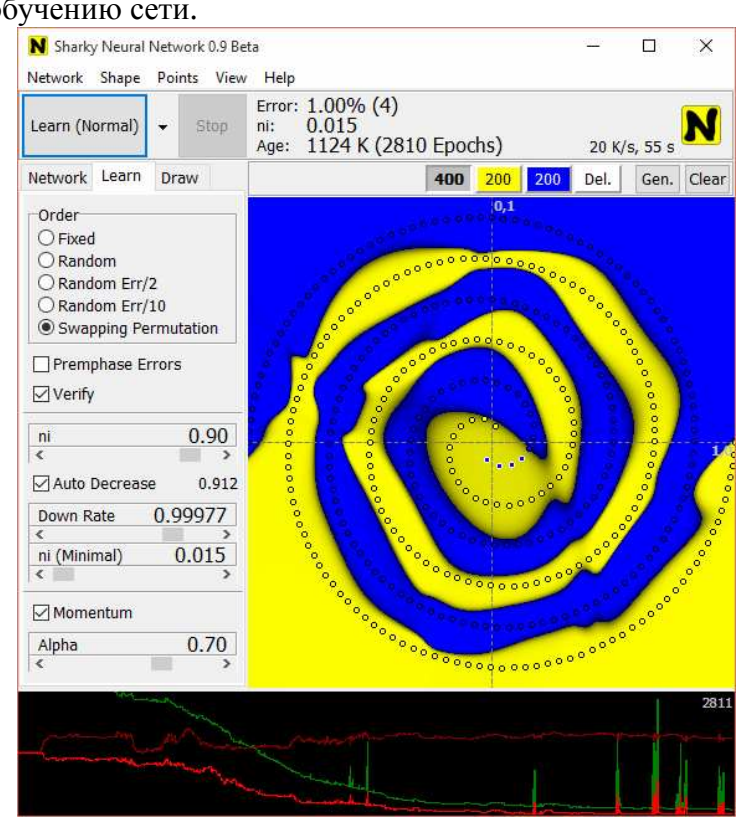


Теперь приступим к обучению. Выбираем четырехслойную сеть с 10 нейронами на каждом из двух скрытых слоев и с 2 нейронами на входе и выходе. (***2:10:10:2***)
Режим обучения оставляем ***Learn(Normal)***.
Далее, все параметры оставляем по умолчанию.



Заданный режим обучения со всеми параметрами и архитектурой сети.

Приступим к обучению сети.



Результат обучения сети

Заметим, что задача была решена за **2810** эпох, при этом процент ошибки составил **1%**. Также мы видим, что числовое значение скорости обучения (ni) составило **0,015**. Под спиралью мы видим трехцветный график. Расшифруем его.

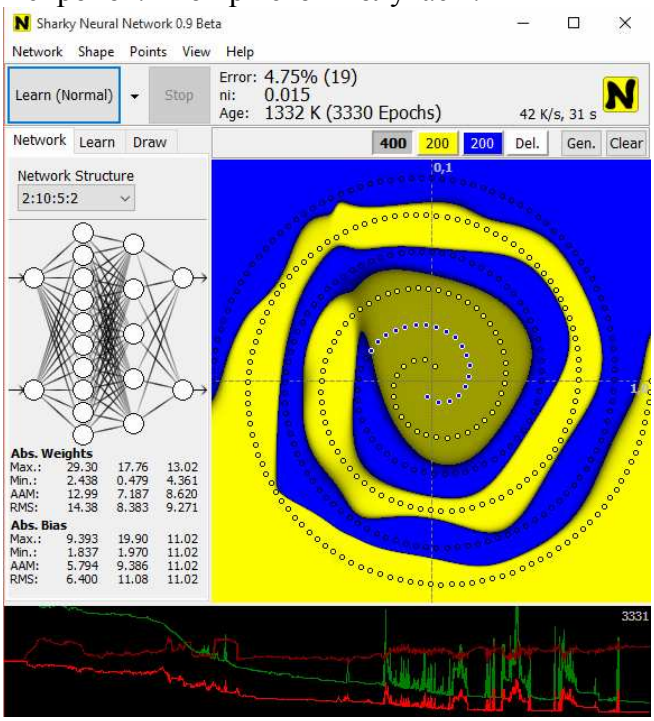
Зеленой линией обозначено значение среднеквадратичной ошибки. Заметим, что это значение убывает, но имеет 7 больших скачков. В эти моменты у сети наблюдалось переобучение. Ярко-красной линией обозначена простая ошибка, как мы видим, она тоже монотонно убывает, имея те же 7 скачков в моменты переобучения. Бордовой линией

обозначена ошибка проверки, которая не имеет значительных скачков и изменений.

Когда наша сеть обучена, приступим к ее контрастированию.

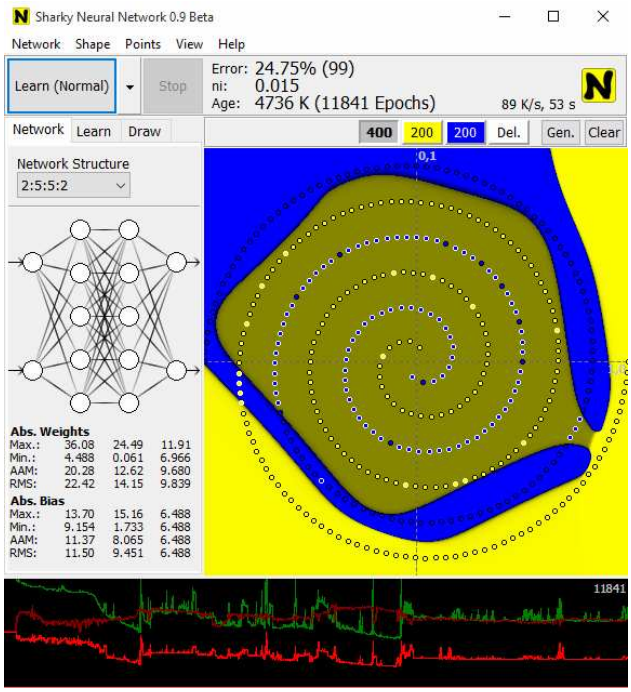
2.Контрастирование обученной сети.

1. Изменим архитектуру сети и рассмотрим получаемые характеристики обучения. Для начала я хочу изменить архитектуру сети, а именно, третий скрытый слой с 10 нейронами заменить на 5 нейронов. Что при этом получаем?



Проводим обучение сети с новой архитектурой. Обращаем внимание, что задача выполнена за **3330** эпох (то есть времени потребовалось больше), причем процент ошибки составляет **4,75%** -что в 4 раза больше предыдущего результата. Также по графикам внизу видно, что к концу обучения возникает явное переобучение, то есть если бы я не нажала ***Stop***, ошибка бы начала увеличиваться еще больше. Числовое значение скорости обучения не изменилось (**0,015**).

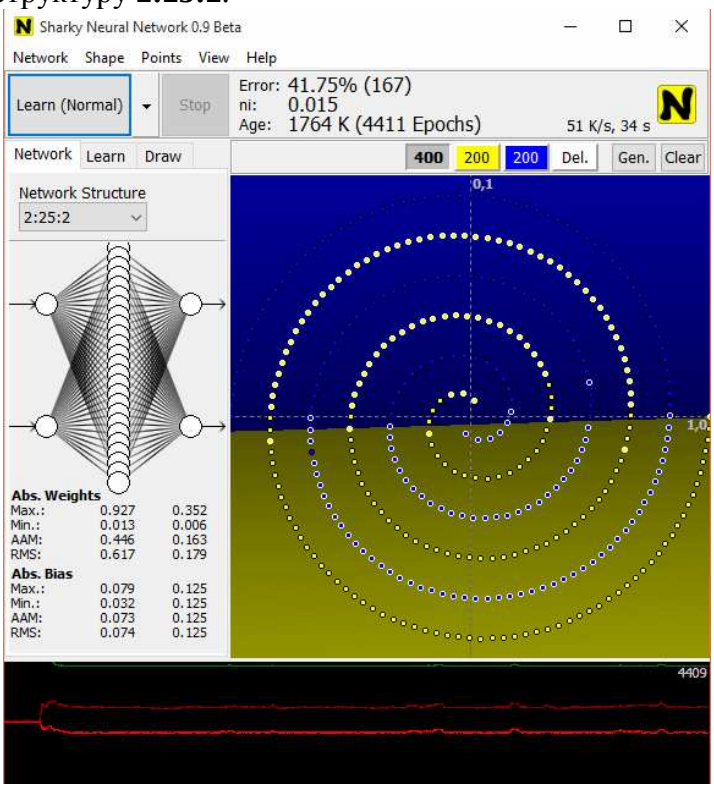
Теперь попробуем уменьшить количество нейронов и во втором, и в третьем слоях, а именно, выберем структуру **2:5:5:2**. Остальные параметры оставим неизменными.



Обучение новой сети

Наблюдаем, что уже такая сеть не справляется с поставленной задачей: на графиках явно видно переобучение, которое ведет к огромной ошибке (мы зафиксировали **25%** на **11841** эпохах, но она может увеличиться еще). Таким образом, уменьшать больше число нейронов нельзя.

Попробуем изменить количество слоев нейронов. Уменьшим наше число слоев до 3. А именно, выберем структуру **2:25:2**.



Обучение нашей сети с 3 слоями

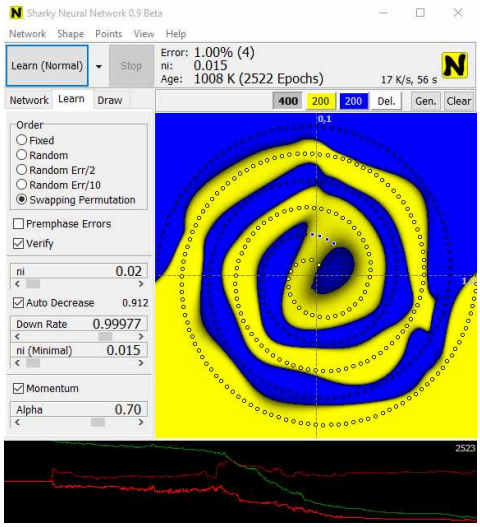
Мы видим, что трехслойная сеть с этой задачей также не справляется. Ошибка колеблется от **40** до **45%** и не уменьшается. При замене 25 нейронов скрытого слоя на 50 и 100 результат не изменяется, процесс лишь замедляется.

Какой вывод мы можем сделать? Минимальная сеть, решающая нашу задачу состоит из **4** слоев, на скрытых слоях которой не меньше **15** нейронов. (т.е. архитектуры

2:10:5:2, **2:10:10:2** справляются).

2. Теперь зададимся вопросом, какое минимальное количество эпох потребуется для выполнения задачи. Так как мы отмели все сети ниже четырехслойных, то нужно определить минимальное число эпох, за которое управятся сети **2:10:10:2** и **2:10:5:2**.

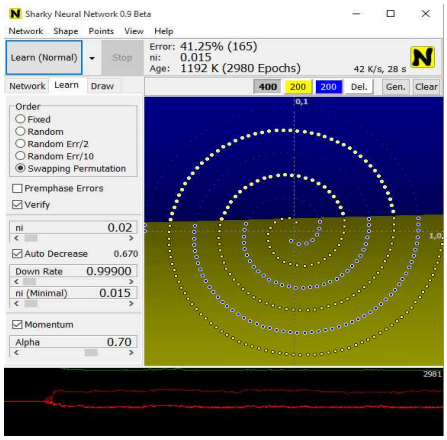
Можно изменить значение **ni**, а именно от **0,90** сделать **0,02**. Рассмотрим архитектуру **2:10:10:2**.



Работа сети 2:10:10:2 с измененным ni

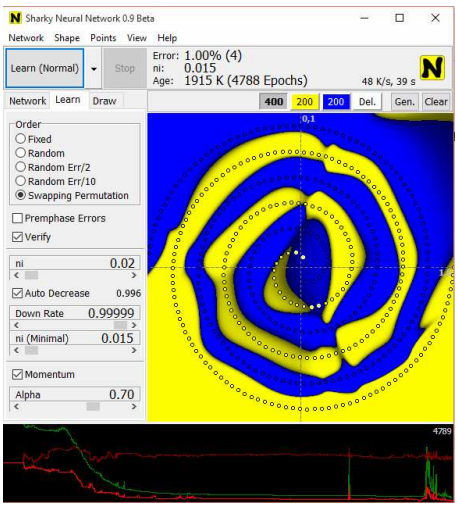
Как видим, количество эпох, за которые задача была выполнена, уменьшилась с **2810** до **2522**.

Если изменить **Down Rate** с **0,99977** до **0,999**, то, по прошествии **2810** эпох, результат не достигается.



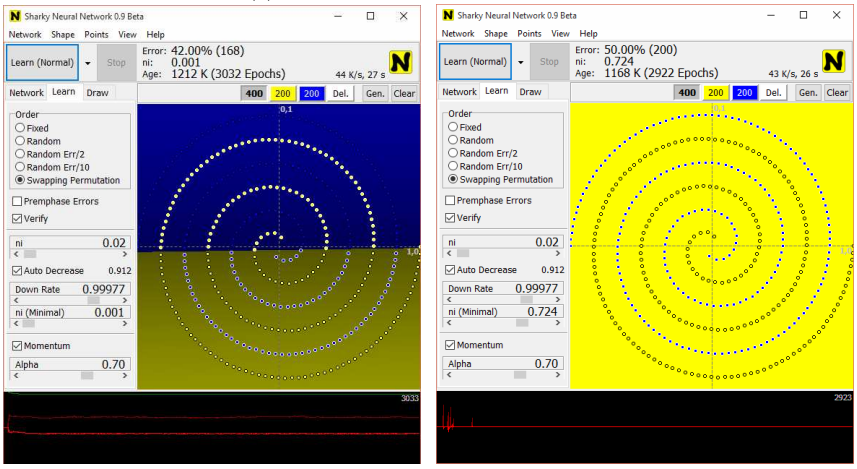
Работа сети с изменением Down Rate

И даже при увеличении этой характеристики до **0,99999**, задача выполняется за большее число эпох (**4788**).



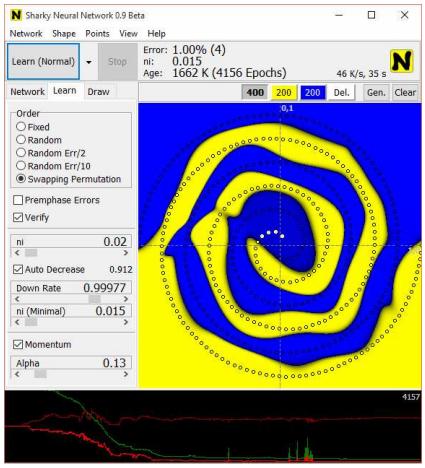
Работа сети с увеличенным Down Rate

При корректировании ni(Minimal) в большую и меньшую стороны наилучшим результатом все равно остается исходный.



Работа сети с измененным ni(Minimal)

Если уменьшить Alpha с **0,70** до **0,13**, то результат будет получен, но тоже за большее число эпох (**4156**).



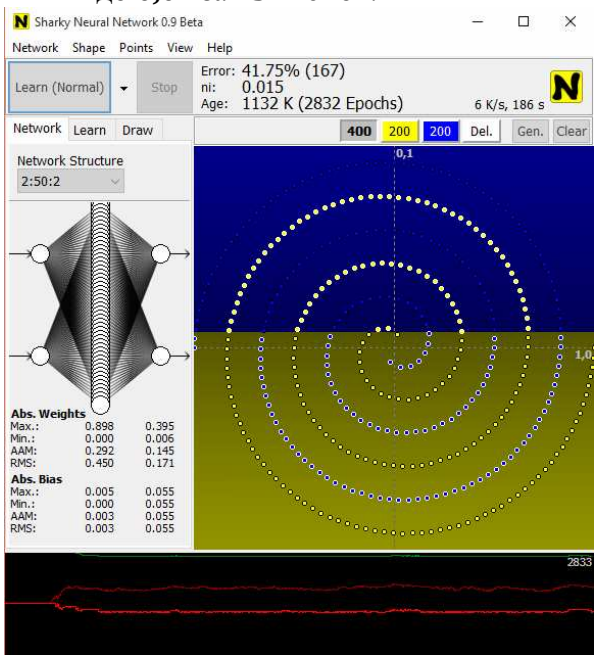
Работа сети с уменьшенным Alpha

Изменив Alpha в большую сторону, мы не сможем решить нашу задачу. Аналогично, поменяв Order нашей сети, результат достигнут не будет.

Таким образом, мы понимаем, что минимальное количество эпох, за которые наша сеть **2:10:10:2** решает поставленную задачу, равно **2522**. Заметим, что это происходит при изменении параметра **ni**, а именно, уменьшении его с **0,90** до **0,02**. При этом изменение других параметров приводит лишь к увеличению количества необходимых эпох или вовсе не решает нашу задачу.

Аналогичную ситуацию мы видим со структурой сети **2:10:5:2**. Только при уменьшении **ni** количество эпох сокращается с **3330** эпох до **3125** эпох.

Можно подумать, что уменьшенное **ni** улучшает работу сети, поэтому, возможно и трехслойная сеть справится с данной задачей? Но, проведя эксперимент, видим, что это не так. Значит, действительно, оптимально с этой задачей справляется четырехслойная сеть вида **2:10:10:2** с уменьшенным **ni** до **0,02** за **2522** эпох.



Работа сети вида 2:50:2 с уменьшенным ni

3.Обобщение изменений работы сети при изменении различных параметров. Переобучение.

Итак, стоит теперь проанализировать ситуацию полностью.

Во-первых, мы выяснили, что, изменяя архитектуру сети, с заданной задачей справляются только две: **2:10:10:2** и **2:10:5:2**. Трехслойные и ниже сети не решают поставленного вопроса.

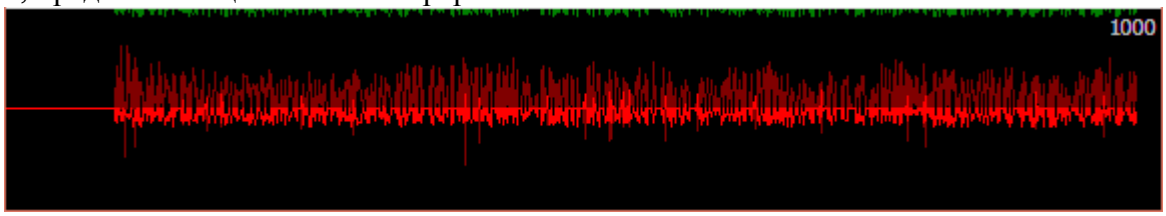
Во-вторых, чтобы оптимизировать время исполнения задачи, необходимо уменьшить **ni** до **0,02**, это приводит к раннему исполнению задачи в обеих сетях.

В-третьих, установление данного параметра не помогает сетям с меньшим числом слоев справиться с задачей.

В-четвертых, при уменьшении **Down Rate** задача не выполняется, а при его увеличении с исходного **0,99977** процесс решения задачи удлиняется. (с обеими сетями)

В-пятых, при любом изменении **ni(Minimal)** наблюдается **ПЕРЕОБУЧЕНИЕ** сети. Это не демонстрируется на приведенных рисунках, т.к. графики внизу слишком быстро скачут, но при наблюдении за процессом работы это видно.

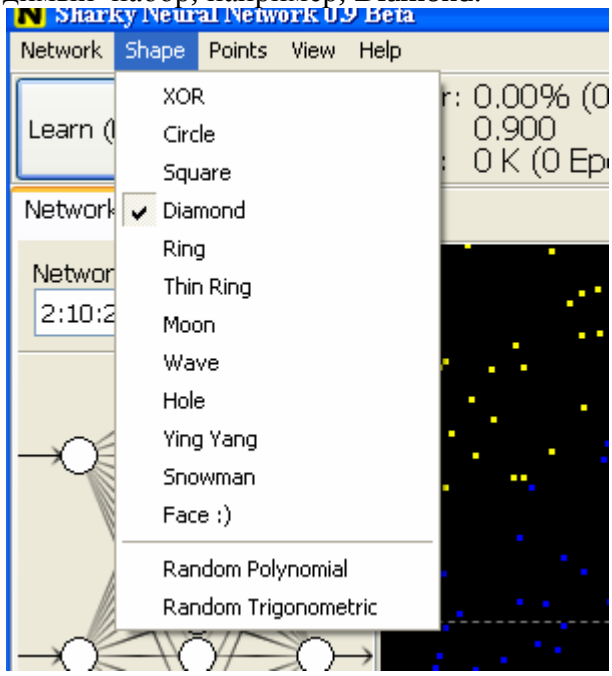
При уменьшении **Alpha (характеристика Momentum)** задача достигается, но за большее число эпох, а при его увеличении также видно **ПЕРЕОБУЧЕНИЕ**. Это видно из графика, представляющего собой непрерывные **скачки**.



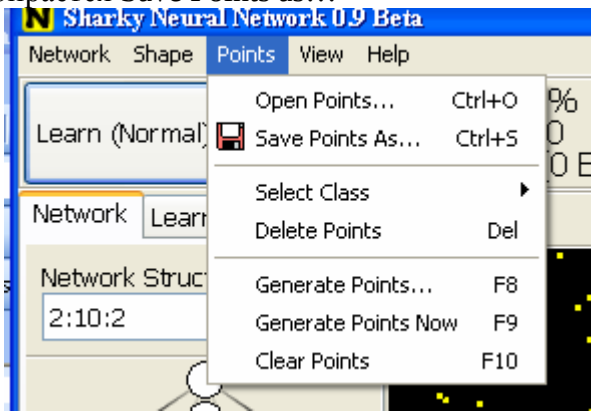
Аналогичную картину наблюдаем при изменении характеристики **Order** с **Swapping Permutation** на любую другую. Т. е. снова **ПЕРЕОБУЧЕНИЕ**.

О загрузке файла из Sharky в Матлаб.

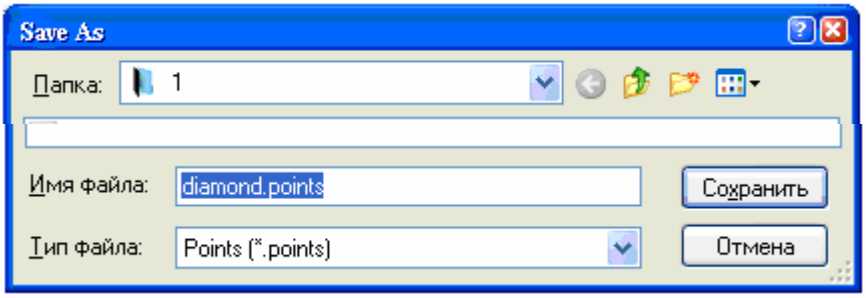
Для загрузки файла, полученного из Shape программы Sharky в Матлаб сначала в меню Shape выделяем необходимый набор, например, Diamond:



Затем в меню Points выбирается Save Points as…



Задаём название папки и имя файла:



После сохранения получаем:

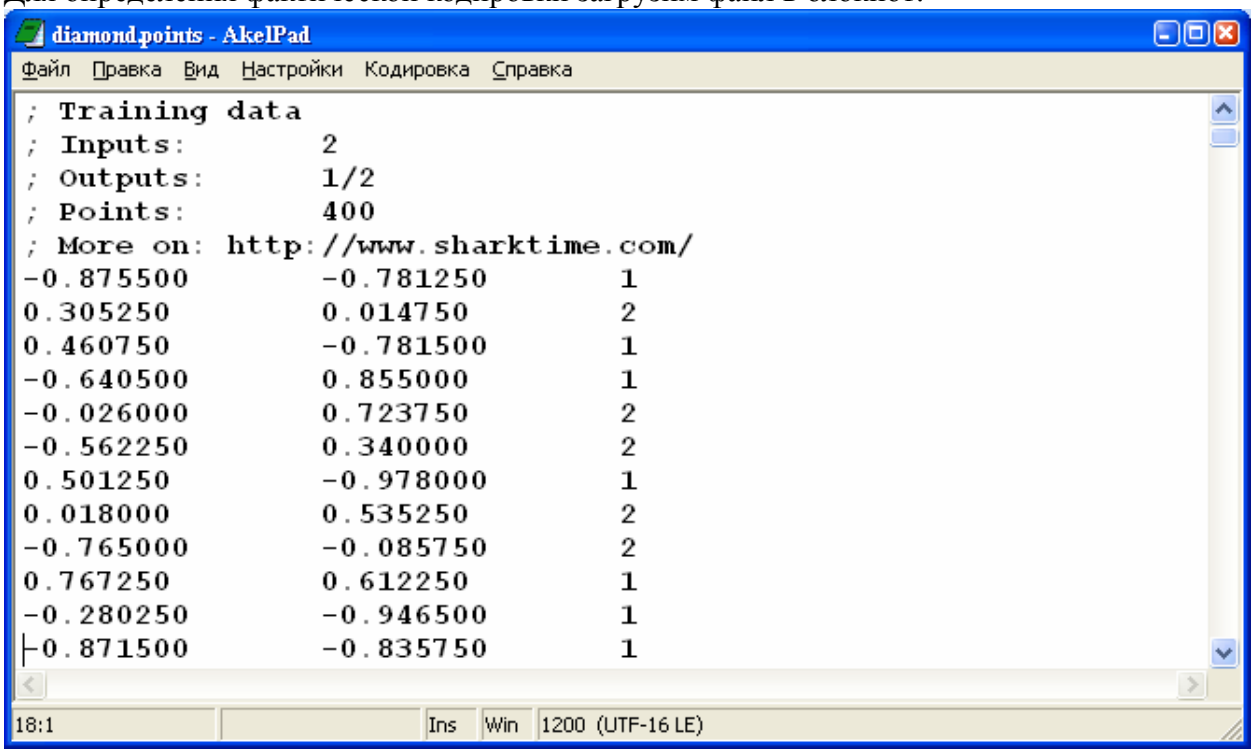
Этот файл содержит:

; Sharky Neural Network 0.9 Beta

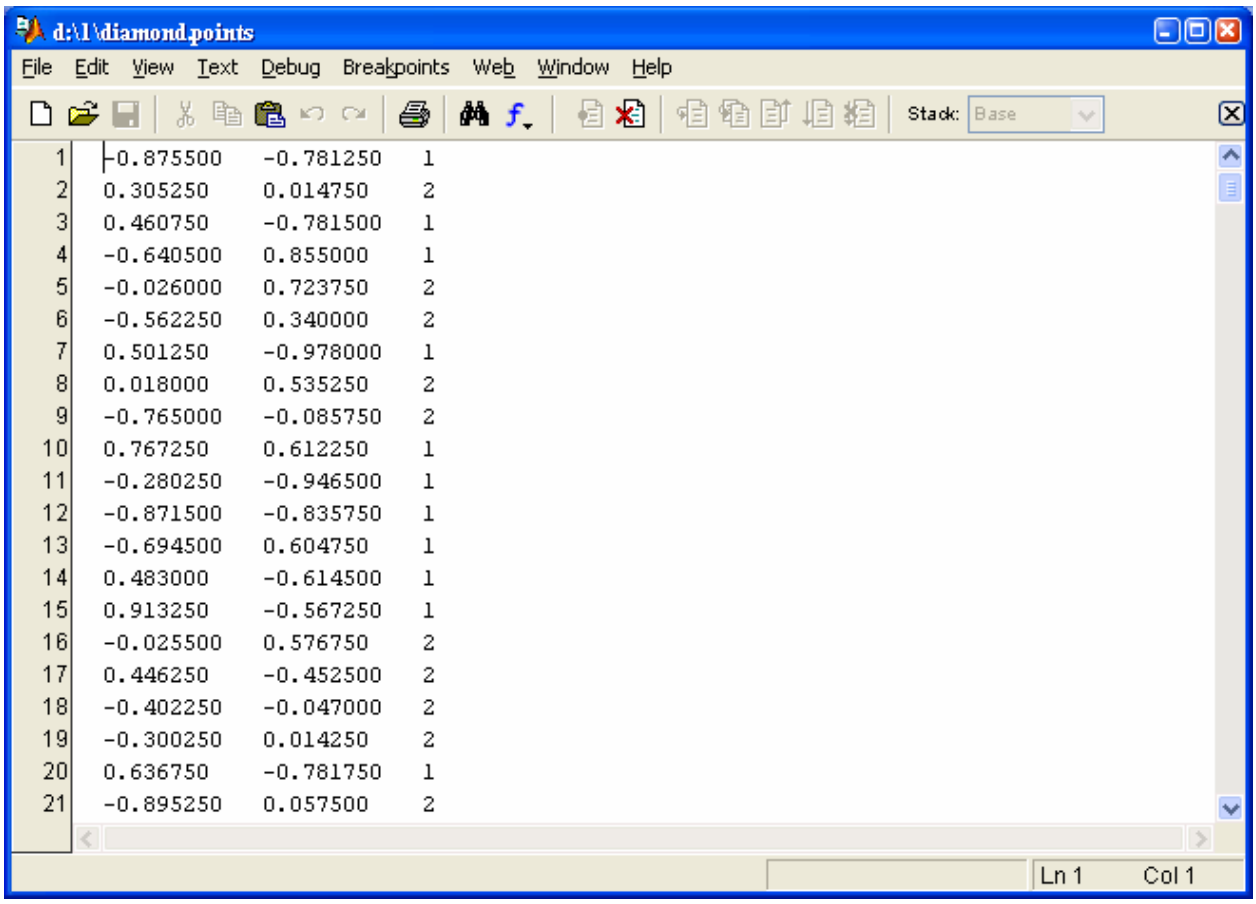
; Training data

; Inputs: 2
; Outputs: 1/2
; Points: 400
; More on: http://www.sharktime.com/
-0.875500 -0.781250 1
0.305250 0.014750 2
0.460750 -0.781500 1
-0.640500 0.855000 1
-0.026000 0.723750 2
-0.562250 0.340000 2
0.501250 -0.978000 1
0.018000 0.535250 2
-0.765000 -0.085750 2
0.767250 0.612250 1
-0.280250 -0.946500 1

Необходимо открыть его и загрузить в переменную «a» в рабочем документе Матлаб. Для этого нужно, чтобы кодировка файла соответствовала требованиям Матлаб : (Ins Win 1251 (ANSI - кириллица). И чтобы файл содержал только однородную информацию, без верхних 5 строк.
Для определения фактической кодировки загрузим файл в блокнот:



В нижней строке читаем текущую кодировку файла: Win 1200 (UTF-16LE).
В меню «Кодировка» выбираем «Сохранить в Windows – 1251».
В блокноте убираем из файла лишние строки.
После этого в Матлаб набираем:
>> open('d:\1\diamond.points') % для открытия файла
>>load('d:\1\diamond.points') % для загрузки файла в рабочую область Матлаб
>>
И получаем результат:



Если при загрузке содержимое файла присвоить переменной “a”, то получим:

```
>> open('d:\1\diamant-1.txt')
>> a=load('d:\1\diamant-1.txt')

a =

   -0.2758   -0.7855    1.0000
   -0.0343   -0.7943    2.0000
    0.2610   -0.5185    2.0000
    0.1630   -0.7815    2.0000
   -0.9547    0.9250    1.0000
    0.5837   -0.6590    1.0000
   -0.2580    0.7823    1.0000
   -0.5062    0.7907    1.0000
   -0.6212    0.3967    1.0000
    0.1328   -0.1213    2.0000
   -0.4108   -0.1948    2.0000
   -0.4725   -0.5430    1.0000
    0.7087    0.5933    1.0000
    0.6445   -0.9520    1.0000
   -0.6820   -0.0135    2.0000
    0.7215   -0.1103    2.0000
        …
```

только файл может оказаться очень длинным, не обязательно его показывать на экране.

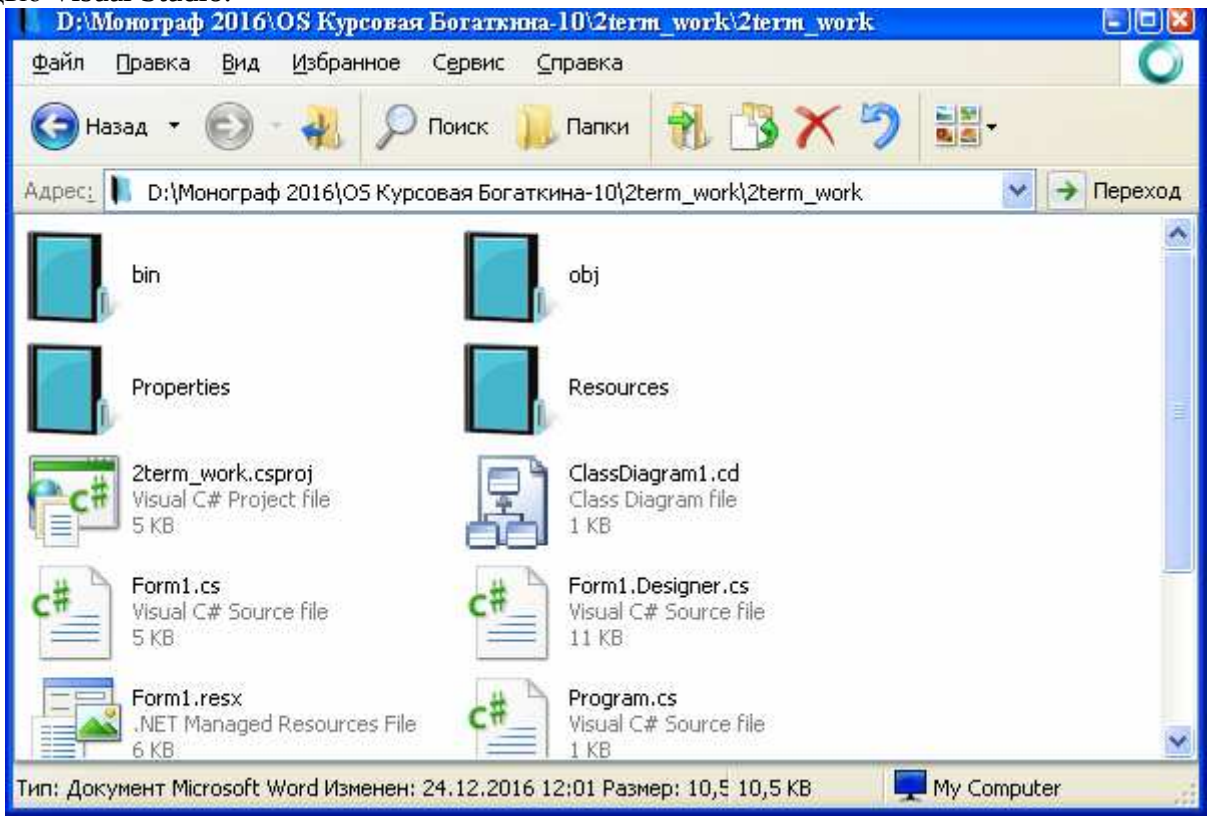
Инструменты для подготовки и проведения нейросетевых исследований

В качестве инструментов, помогающих в подготовке и проведении нейросетевых исследований можно рассмотреть программу объединения исходных текстов многомодульного программного проекта Visual Studio в один файл; Анализатор исходного кода C#-программы (CSharpAnalysis); Анализатор длины файлов в папке и Программу для проверки папок на наличие длинных имён.

Программа объединения исходных_текстов-модулей проекта Visual Studio в один файл

При использовании больших пространств имён для выполнения конкретного нейросетевого исследования необходимо собрать нейронную сеть требуемой конфигурации из имеющихся программ.

Такая задача возникает и при анализе программных проектов, выполненных с помощью Visual Studio.



Программный проект Visual Studio содержит большое количество файлов и папок, среди которых только несколько (в рассматриваемом примере - три файла) содержат исходные тексты: Form1.cs, Form1.Designer.cs и Program.cs. Преобразование многофайлового проекта Visual Studio в единственный исходный cs-файл предусматривает следующую последовательность действий:

1. Файлы с исходными текстами выписываются последовательно в текстовый файл. Если из текстового файла перенести в файл Word и проставить номера строк, можно будет легко излагать технологию преобразования многофайловой конструкции в один исходный cs-файл.
2. Полученный набор файлов необходимо отредактировать с учётом требований языка C# (указание используемых ресурсов, границы пространств имён в каждом файле, границы классов, и т.д.).

3. Проверка результатов редактирования может проводиться компилятором CSC.

4. Заканчивается проверка компиляцией cs-файла и получением исполнимого exe-файла (реже – получением dll-файла).

5. Запуск программы.

В рассматриваемом примере многофайловая конструкция занимает 286 строк (операторов).

Файл Form1 занимает с 1 по 69 строки.

Файл Form1.Designer.cs занимает с 70 по 265 строки.

Файл Program.cs занимает с 266 строки по 286.

Со 2 по 11 строку содержатся перечисления подключаемых пространств имён. Такие же операторы могут встречаться в начале каждого файла. Но у Form1.Designer.cs их нет. Тогда как в файле Program.cs они занимают с 267 по 270 строки. Их надо переместить к 12 строке и проверить на повторяемость. Повторяющиеся операторы удаляются.

Поскольку в нашем случае все 4 оператора уже содержатся среди 2-11, строки 267-270 просто удаляются.

Нужно обратить внимание на 71-74 строки (переход от 1 файла ко 2). Они лишние. Их можно удалить. И удалить 69 оператор с разделителем 1 и 2 классов. Таким образом, фигурная скобка, закрывающая первый файл и начало 2 файла удаляются, соединив первые 2 файла в один – вместо разделённого (partial) класса создаётся единственный класс Form1.

69. }
70. //Form1.Designer.cs
71. namespace _2term_work
72. {
73. partial class Form1
74. {

Аналогично нужно соединить 2 и 3 файлы, но при их соединении границы классов должны быть соблюдены. Удаляются только 265, 271 и 272 строки.

Текстовый файл переименовываем в z.cs и начинается проверка его правильности компилятором csc.

Единый исходный файл z.cs после всех описанных преобразований содержит:

```
//form1.cs
using System;
using System.IO;
using System.Diagnostics;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace _2term_work
{
    public partial class Form1 : Form
    {
        string[] a;
        string str;
        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            if (textBox1.Text.Equals("")) { MessageBox.Show("Выберите каталог
для проверки"); }
            else
            {
                a = new string[100];

                try
                {
                    string[] subdir = Directory.GetDirectories(textBox1.Text,
"*", System.IO.SearchOption.AllDirectories);
                    int k = 0;
                    int j = 0;
                    for (int i = 0; i < subdir.Length; i++)
                    {
                        if (subdir[i].Length > 200)
                        {
                            label1.Text = "В папке обнаружены имена длиной
более 200 символов";
                            k += 1;
                            a[j] = subdir[i]; j++;
                        }
                    }
                    listBox1.DataSource = a;
                    if (k == 0)
                    {
                        label1.Text = "Папок с длиной имени более 200
символов не обнаружено";
                    }
                }
                catch { MessageBox.Show("Выбранная папка содержит защищенные
каталоги"); }
            }
        }
        private void button2_Click(object sender, EventArgs e)
        {
            FolderBrowserDialog d = new FolderBrowserDialog();
            d.Description = "Выберите папку для проверки";
            d.SelectedPath = @"c:\";
            if (d.ShowDialog() == DialogResult.OK)
            {
                textBox1.Text = d.SelectedPath;
            }
        }
        private void button3_Click(object sender, EventArgs e)
        {
            string dir = "", dir1 = "";
            if (listBox1.SelectedIndex == -1) { MessageBox.Show("Выберите в
списке слева папку для реконструкции"); }
            else
            {
                int k = listBox1.SelectedIndex;
                string[] d = a[k].Split('\\');
                string[] g = textBox1.Text.Split('\\');
                dir = textBox1.Text + "\\" + d[g.Length];
                dir1 = dir + ".rar";
                Process pro = new Process();
                pro.StartInfo.CreateNoWindow = true;
                pro.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
                pro.StartInfo.Arguments = "a -y -inul -ibck " + dir1 + " " +
dir;
                pro.StartInfo.FileName = "WinRAR";
                pro.Start();
```

```
                str = dir;
            }
        }
        private void button4_Click(object sender, EventArgs e)
        {
            Dispose();
        }
        private void button5_Click(object sender, EventArgs e)
        {
            int k = 0;
            Directory.Delete(str, true);
            str = "";
            a = new string[100];
            listBox1.DataSource = a;
            string[] subdir = Directory.GetDirectories(textBox1.Text, "*",
System.IO.SearchOption.AllDirectories);
            int p = 0;
            int j = 0;

            for (int i = 0; i < subdir.Length; i++)
            {
                if (subdir[i].Length > 200)
                {
                    label1.Text = "В папке обнаружены имена длиной более 200
символов!";
                    p += 1;
                    a[j] = subdir[i]; j++;
                }
            }
            listBox1.DataSource = a;
            if (k == 0)
            {
                label1.Text = "Папок с длиной имени более 200 символов не
обнаружено";
            }
        }
//Form1.Designer.cs
        /// <summary>
        /// Требуется переменная конструктора.
        /// </summary>
        private System.ComponentModel.IContainer components = null;
        /// <summary>
        /// Освободить все используемые ресурсы.
        /// </summary>
        /// <param name="disposing">истинно, если управляемый ресурс должен
быть удален; иначе ложно.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
        #region Код, автоматически созданный конструктором форм Windows
        /// <summary>
        /// Обязательный метод для поддержки конструктора - не изменяйте
        /// содержимое данного метода при помощи редактора кода.
        /// </summary>
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.button2 = new System.Windows.Forms.Button();
```

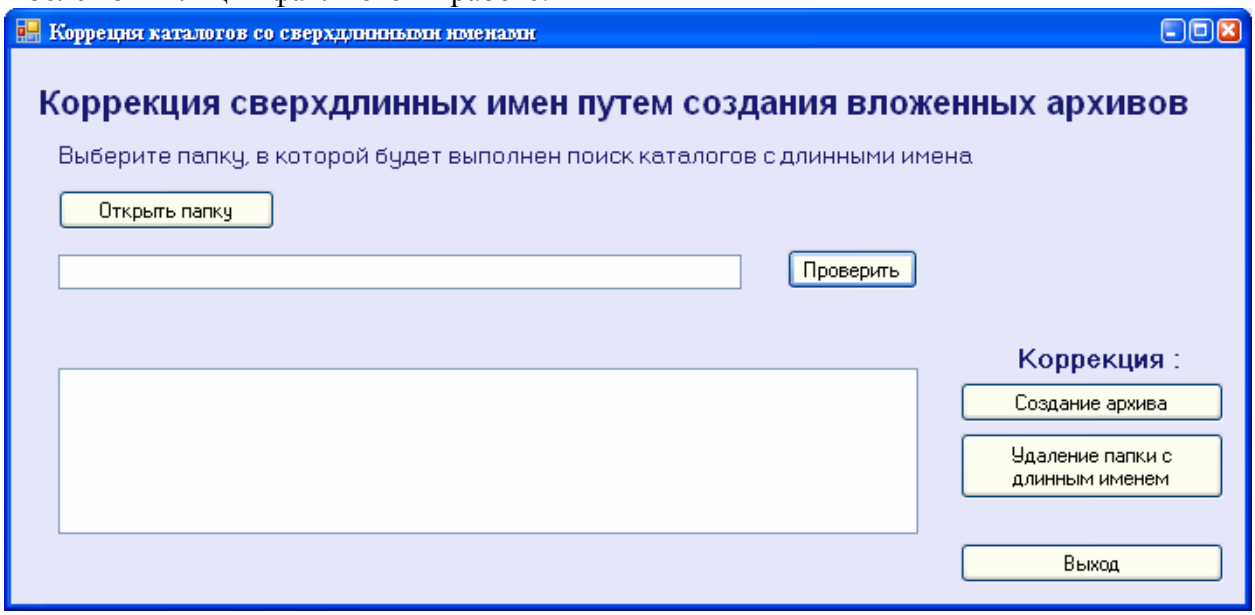
```
            this.folderBrowserDialog1 = new
System.Windows.Forms.FolderBrowserDialog();
            this.label1 = new System.Windows.Forms.Label();
            this.listBox1 = new System.Windows.Forms.ListBox();
            this.button3 = new System.Windows.Forms.Button();
            this.label3 = new System.Windows.Forms.Label();
            this.button4 = new System.Windows.Forms.Button();
            this.label2 = new System.Windows.Forms.Label();
            this.button5 = new System.Windows.Forms.Button();
            this.label4 = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.BackColor = System.Drawing.Color.Ivory;
            this.button1.Location = new System.Drawing.Point(444, 115);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(75, 23);
            this.button1.TabIndex = 0;
            this.button1.Text = "Проверить";
            this.button1.UseVisualStyleBackColor = false;
            this.button1.Click += new
System.EventHandler(this.button1_Click);
            //
            // textBox1
            //
            this.textBox1.Location = new System.Drawing.Point(27, 118);
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(391, 20);
            this.textBox1.TabIndex = 1;
            //
            // button2
            //
            this.button2.BackColor = System.Drawing.Color.Ivory;
            this.button2.Font = new System.Drawing.Font("Microsoft Sans
Serif", 8.25F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.button2.Location = new System.Drawing.Point(27, 81);
            this.button2.Name = "button2";
            this.button2.Size = new System.Drawing.Size(124, 23);
            this.button2.TabIndex = 2;
            this.button2.Text = "Открыть папку";
            this.button2.UseVisualStyleBackColor = false;
            this.button2.Click += new
System.EventHandler(this.button2_Click);
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Font = new System.Drawing.Font("Palatino Linotype",
11.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.label1.ForeColor = System.Drawing.Color.MidnightBlue;
            this.label1.Location = new System.Drawing.Point(23, 146);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(0, 20);
            this.label1.TabIndex = 3;
            //
            // listBox1
            //
            this.listBox1.FormattingEnabled = true;
            this.listBox1.HorizontalScrollbar = true;
            this.listBox1.Location = new System.Drawing.Point(27, 183);
            this.listBox1.Name = "listBox1";
```

```
            this.listBox1.Size = new System.Drawing.Size(492, 95);
            this.listBox1.TabIndex = 4;
            //
            // button3
            //
            this.button3.BackColor = System.Drawing.Color.Ivory;
            this.button3.Location = new System.Drawing.Point(543, 191);
            this.button3.Name = "button3";
            this.button3.Size = new System.Drawing.Size(151, 23);
            this.button3.TabIndex = 5;
            this.button3.Text = "Создание архива";
            this.button3.UseVisualStyleBackColor = false;
            this.button3.Click += new
System.EventHandler(this.button3_Click);
            //
            // label3
            //
            this.label3.AutoSize = true;
            this.label3.BackColor = System.Drawing.Color.Lavender;
            this.label3.Font = new System.Drawing.Font("Microsoft Sans
Serif", 9.75F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.label3.ForeColor = System.Drawing.Color.MidnightBlue;
            this.label3.Location = new System.Drawing.Point(24, 53);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(531, 16);
            this.label3.TabIndex = 8;
            this.label3.Text = "Выберите папку, в которой будет выполнен
поиск каталогов с длинными имена";
            //
            // button4
            //
            this.button4.BackColor = System.Drawing.Color.Ivory;
            this.button4.Location = new System.Drawing.Point(543, 283);
            this.button4.Name = "button4";
            this.button4.Size = new System.Drawing.Size(151, 23);
            this.button4.TabIndex = 9;
            this.button4.Text = "Выход";
            this.button4.UseVisualStyleBackColor = false;
            this.button4.Click += new
System.EventHandler(this.button4_Click);
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Font = new System.Drawing.Font("Arial", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.label2.ForeColor = System.Drawing.Color.MidnightBlue;
            this.label2.Location = new System.Drawing.Point(12, 20);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(668, 22);
            this.label2.TabIndex = 10;
            this.label2.Text = "Коррекция сверхдлинных имен путем создания
вложенных архивов";
            //
            // button5
            //
            this.button5.BackColor = System.Drawing.Color.Ivory;
            this.button5.Location = new System.Drawing.Point(543, 220);
            this.button5.Name = "button5";
            this.button5.Size = new System.Drawing.Size(151, 38);
            this.button5.TabIndex = 11;
            this.button5.Text = "Удаление папки с длинным именем";
```

```
            this.button5.UseVisualStyleBackColor = false;
            this.button5.Click += new
System.EventHandler(this.button5_Click);
            //
            // label4
            //
            this.label4.AutoSize = true;
            this.label4.Font = new System.Drawing.Font("Microsoft Sans
Serif", 11.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.label4.ForeColor = System.Drawing.Color.MidnightBlue;
            this.label4.Location = new System.Drawing.Point(573, 168);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(102, 18);
            this.label4.TabIndex = 12;
            this.label4.Text = "Коррекция :";
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackColor = System.Drawing.Color.Lavender;
            this.ClientSize = new System.Drawing.Size(706, 318);
            this.Controls.Add(this.label4);
            this.Controls.Add(this.button5);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.button4);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.button3);
            this.Controls.Add(this.listBox1);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.button2);
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.button1);
            this.Font = new System.Drawing.Font("Microsoft Sans Serif",
8.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.Name = "Form1";
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Корреция каталогов со сверхдлинными именами";
            this.ResumeLayout(false);
            this.PerformLayout();
        }
        #endregion
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.FolderBrowserDialog
folderBrowserDialog1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.ListBox listBox1;
        private System.Windows.Forms.Button button3;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Button button4;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Button button5;
        private System.Windows.Forms.Label label4;
    }
//Program.cs
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
```

```
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

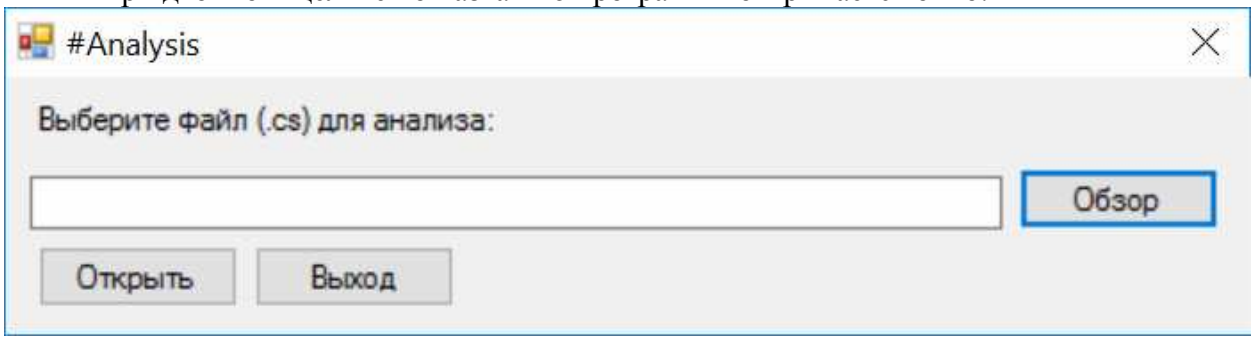
После компиляции файл готов к работе:



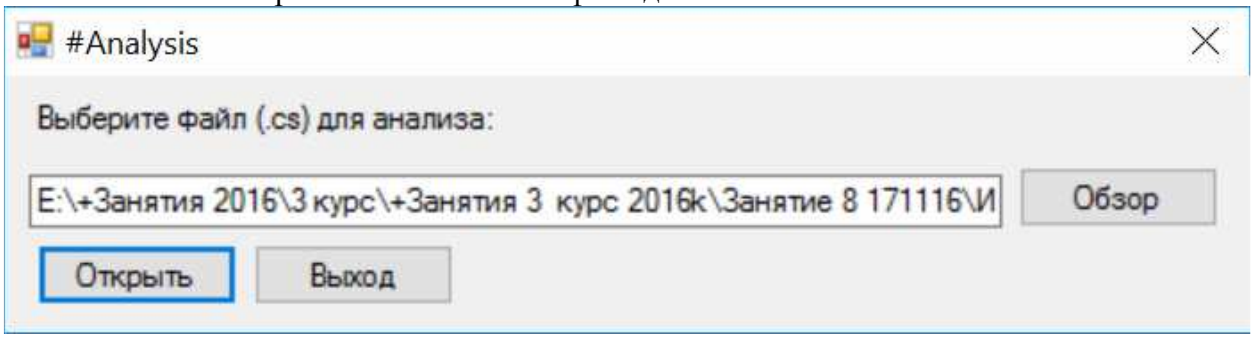
Исходный текст файла может быть легко изменён для коррекции предусмотренных в нём функций.

Анализатор исходного кода C#-программы (CSharpAnalysis).

При двойном щелчке по названию программы открывается окно:

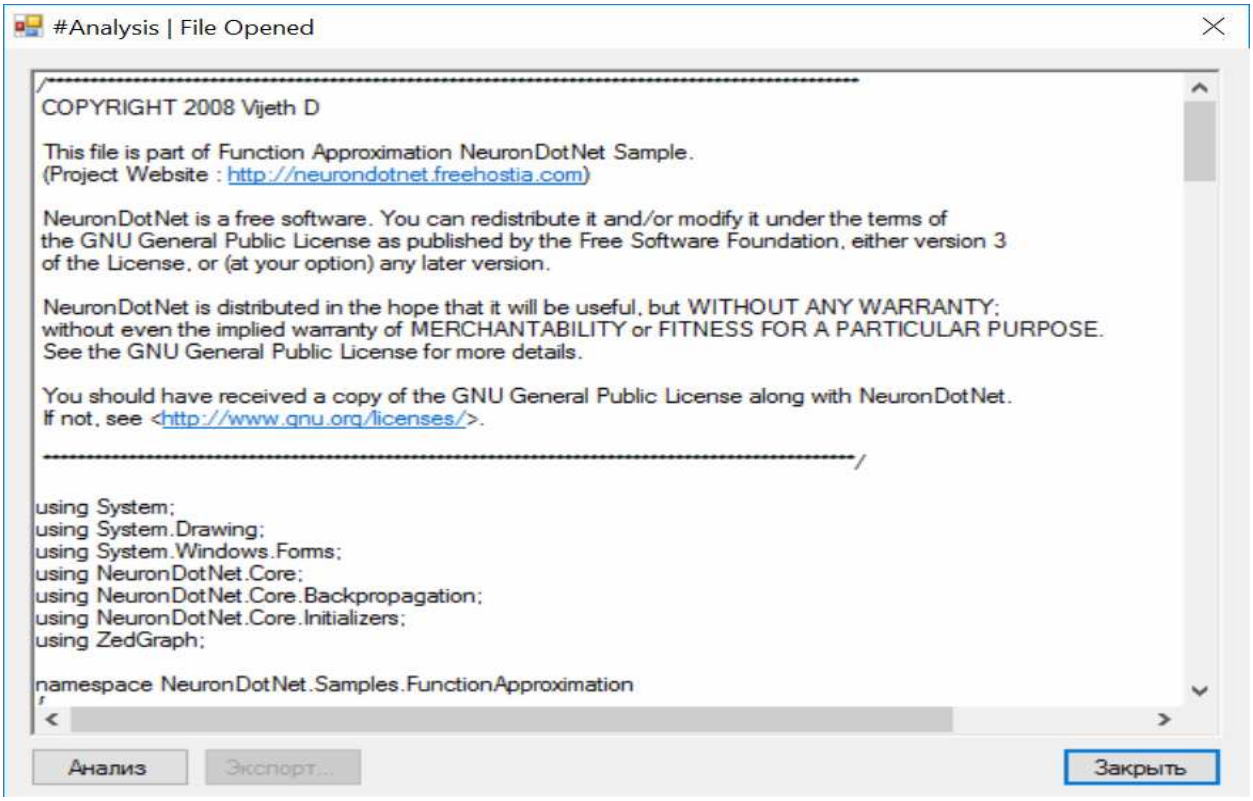


Кнопка Обзор позволяет найти cs-файл для анализа

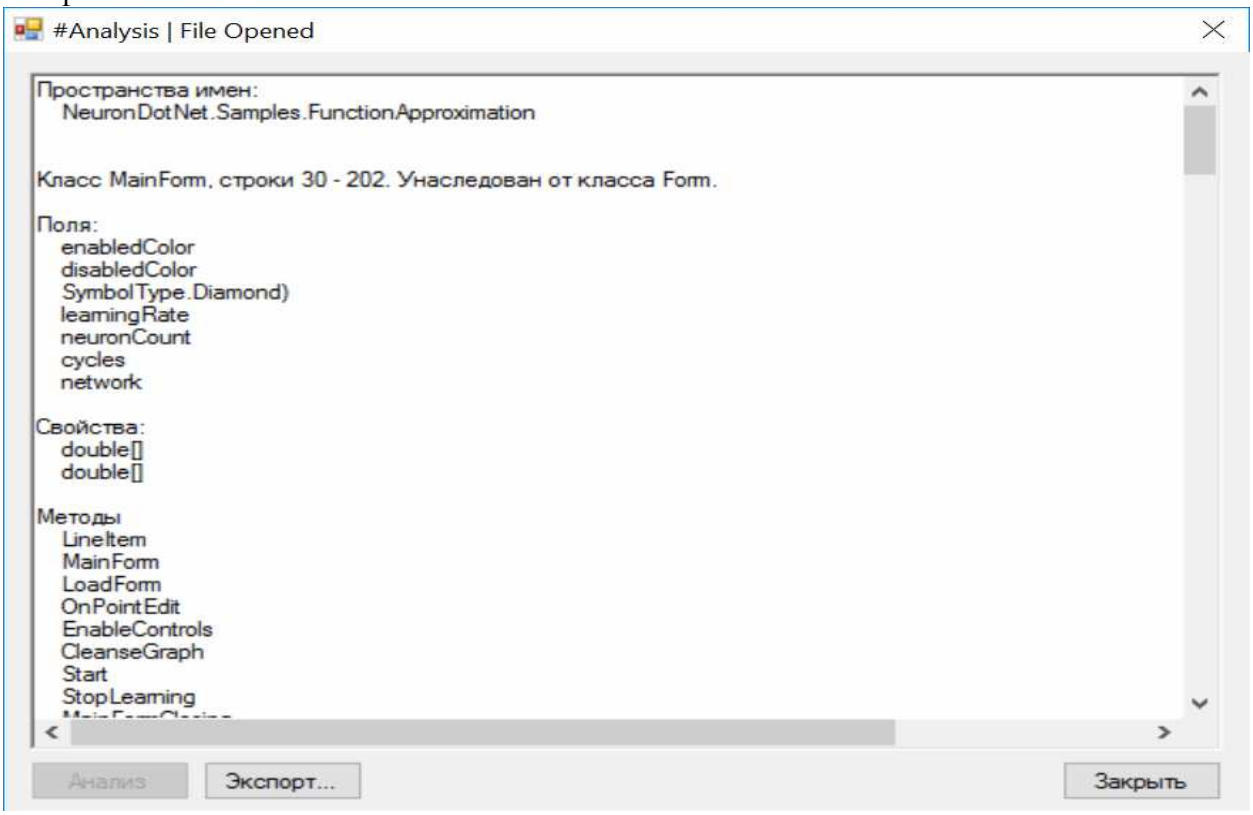


и загрузить его по кнопке Открыть. При этом изменяется окно и становится возможным

просмотр файла:



После нажатия «Анализ» появляется результат анализа, за которым следует текст исходного файла:



В результате анализа из исходного текста выбираются:

- Используемые пространства имён;
- Классы с указанием их границ и наследственной иерархии;
- Используемые поля;

- Свойства;
- Методы.

Выбранная информация оформляется в виде:

```
Пространства имен:
   NeuronDotNet.Samples.FunctionApproximation
Класс MainForm, строки 30 - 202. Унаследован от класса Form.
Поля:
   enabledColor
   disabledColor
   SymbolType.Diamond)
   learningRate
   neuronCount
   cycles
   network
Свойства:
   double[]
   double[]
Методы
   LineItem
   MainForm
   LoadForm
   OnPointEdit
   EnableControls
   CleanseGraph
   Start
   StopLearning
   MainFormClosing
```

После вывода аналитической информации следует программа исследуемого cs-файла с номерами строк. При больших размерах программ это существенно облегчает аналитические операции:

```
001
/*************************************************************************
002   COPYRIGHT 2008 Vijeth D
003
004   This file is part of Function Approximation NeuronDotNet Sample.
005   (Project Website : http://neurondotnet.freehostia.com)
006
007   NeuronDotNet is a free software. You can redistribute it and/or modify
it under the terms of
008   the GNU General Public License as published by the Free Software
Foundation, either version 3
009   of the License, or (at your option) any later version.
010
011   NeuronDotNet is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY;
012   without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
013   See the GNU General Public License for more details.
014
015   You should have received a copy of the GNU General Public License along
with NeuronDotNet.
016   If not, see <http://www.gnu.org/licenses/>.
017
018
*************************************************************************
*/
019
020  using System;
021  using System.Drawing;
022  using System.Windows.Forms;
023  using NeuronDotNet.Core;
```

```
024  using NeuronDotNet.Core.Backpropagation;
025  using NeuronDotNet.Core.Initializers;
026  using ZedGraph;
027
028  namespace NeuronDotNet.Samples.FunctionApproximation
029  {
030      public partial class MainForm : Form
031      {
032          private static readonly Color enabledColor = Color.Tomato;
033          private static readonly Color disabledColor = Color.Goldenrod;
034
035          private LineItem curve = new LineItem("",
036              new double[] { 1, 2, 3, 4, 5, 6 },
037              new double[] { 0.01, 0.03, 0.07, 0.15, 0.31, 0.63},
038              enabledColor,
039              SymbolType.Diamond);
040
041          private double learningRate = 0.3d;
042          private int neuronCount = 10;
043          private int cycles = 10000;
044          private BackpropagationNetwork network;
045
046          public MainForm()
047          {
048              InitializeComponent();
049          }
050
051          private void LoadForm(object sender, EventArgs e)
052          {
053              GraphPane pane = functionGraph.GraphPane;
054              pane.Chart.Fill = new Fill(Color.AntiqueWhite,
Color.Honeydew, -45F);
055              pane.Title.IsVisible = false;
056              pane.YAxis.IsVisible = false;
057              pane.YAxis.Scale.Max = 1d;
058              pane.YAxis.Scale.Min = 0d;
059
060              pane.Legend.IsVisible = false;
061
062              pane.XAxis.Title.Text = "Drag the points to position them
and define the function";
063              pane.XAxis.Title.FontSpec.Size = 15;
064              pane.XAxis.MajorGrid.IsVisible = true;
065              pane.XAxis.Scale.IsVisible = false;
066
067              curve.Line.IsVisible = false;
068              curve.Symbol.Fill.Type = FillType.Solid;
069              functionGraph.ContextMenuBuilder += new
ZedGraphControl.ContextMenuBuilderEventHandler(
070                  delegate(ZedGraphControl graph, ContextMenuStrip
menuStrip, Point mousePt, ZedGraphControl.ContextMenuObjectState objState)
071                  {
072                      for (int i = 0; i < menuStrip.Items.Count; i++)
073                      {
074                          string tag = (string)menuStrip.Items[i].Tag;
075                          if (tag == "undo_all" || tag == "unzoom" || tag
== "set_default")
076                          {
077                              menuStrip.Items.RemoveAt(i);
078                              i--;
079                          }
080                      }
081                  });
082
```

```
083             functionGraph.IsEnableVEdit = true;
084             functionGraph.EditModifierKeys = Keys.None;
085             functionGraph.EditButtons = MouseButtons.Left;
086             functionGraph.IsEnableZoom = false;
087             functionGraph.IsEnableSelection = false;
088             functionGraph.IsEnableHPan = false;
089             functionGraph.IsEnableVPan = false;
090             functionGraph.GraphPane.CurveList.Capacity = 2;
091             OnPointEdit(functionGraph, pane, curve, 0);
092             pane.AxisChange();
093             EnableControls(true);
094             txtCycles.Text = cycles.ToString();
095             txtLearningRate.Text = learningRate.ToString();
096             txtNeuronCount.Text = neuronCount.ToString();
097         }
098
099         private string OnPointEdit(ZedGraphControl sender, GraphPane
pane, CurveItem curve, int iPt)
100         {
101             listData.Items.Clear();
102             for (int i = 0; i < this.curve.Points.Count; i++)
103             {
104                 listData.Items.Add((i + 1).ToString());
105                 this.curve.Points[i].Y = Math.Max(0.01, Math.Min(0.99,
this.curve.Points[i].Y));
106
listData.Items[i].SubItems.Add(this.curve.Points[i].Y.ToString("0.00"));
107             }
108             CleanseGraph();
109             return null;
110         }
111
112         private void EnableControls(bool enabled)
113         {
114             btnStart.Enabled = enabled;
115             btnStop.Enabled = !enabled;
116             txtCycles.Enabled = enabled;
117             txtLearningRate.Enabled = enabled;
118             txtNeuronCount.Enabled = enabled;
119             trainingProgressBar.Value = 0;
120
121             functionGraph.IsEnableVEdit = enabled;
122             curve.Color = enabled ? enabledColor : disabledColor;
123         }
124
125         private void CleanseGraph()
126         {
127             functionGraph.GraphPane.CurveList.Clear();
128             functionGraph.GraphPane.CurveList.Add(curve);
129             functionGraph.Invalidate();
130         }
131
132         private void Start(object sender, EventArgs e)
133         {
134             CleanseGraph();
135             EnableControls(false);
136             curve.Color = enabledColor;
137
138             if (!int.TryParse(txtCycles.Text, out cycles)) { cycles =
10000; }
139             if (!double.TryParse(txtLearningRate.Text, out
learningRate)) { learningRate = 0.25d; }
140             if (!int.TryParse(txtNeuronCount.Text, out neuronCount)) {
neuronCount = 10; }
```

```
141
142             if (cycles <= 0) { cycles = 10000; }
143             if (learningRate < 0 || learningRate > 1) { learningRate =
0.25d; }
144             if (neuronCount <= 0) { neuronCount = 10; }
145
146             txtCycles.Text = cycles.ToString();
147             txtLearningRate.Text = learningRate.ToString();
148             txtNeuronCount.Text = neuronCount.ToString();
149
150             LinearLayer inputLayer = new LinearLayer(1);
151             SigmoidLayer hiddenLayer = new SigmoidLayer(neuronCount);
152             SigmoidLayer outputLayer = new SigmoidLayer(1);
153             new BackpropagationConnector(inputLayer,
hiddenLayer).Initializer = new RandomFunction(0d, 0.3d);
154             new BackpropagationConnector(hiddenLayer,
outputLayer).Initializer = new RandomFunction(0d, 0.3d);
155             network = new BackpropagationNetwork(inputLayer,
outputLayer);
156             network.SetLearningRate(learningRate);
157
158             TrainingSet trainingSet = new TrainingSet(1, 1);
159             for (int i = 0; i < curve.Points.Count; i++)
160             {
161                 double xVal = curve.Points[i].X;
162                 for (double input = xVal - 0.05; input < xVal + 0.06;
input += 0.01)
163                 {
164                     trainingSet.Add(new TrainingSample(new double[] {
input }, new double[] { curve.Points[i].Y}));
165                 }
166             }
167
168             network.EndEpochEvent += new TrainingEpochEventHandler(
169                 delegate(object senderNetwork, TrainingEpochEventArgs
args)
170                 {
171                     trainingProgressBar.Value =
(int)(args.TrainingIteration * 100d / cycles);
172                     Application.DoEvents();
173                 });
174             network.Learn(trainingSet, cycles);
175             StopLearning(this, EventArgs.Empty);
176         }
177
178         private void StopLearning(object sender, EventArgs e)
179         {
180             if (network != null)
181             {
182                 network.StopLearning();
183                 LineItem lineItem = new LineItem("Approximated
Function");
184                 for (double xVal = 0; xVal < 10; xVal += 0.05d)
185                 {
186                     lineItem.AddPoint(xVal, network.Run(new double[] {
xVal })[0]);
187                 }
188                 lineItem.Symbol.Type = SymbolType.None;
189                 lineItem.Color = Color.DarkOrchid;
190                 functionGraph.GraphPane.CurveList.Add(lineItem);
191                 functionGraph.Refresh();
192                 functionGraph.GraphPane.CurveList.Remove(lineItem);
193             }
194             network = null;
```

```
195             EnableControls(true);
196         }
197
198         private void MainFormClosing(object sender, FormClosingEventArgs
e)
199         {
200             StopLearning(this, EventArgs.Empty);
201         }
202     }
203 }
204
```

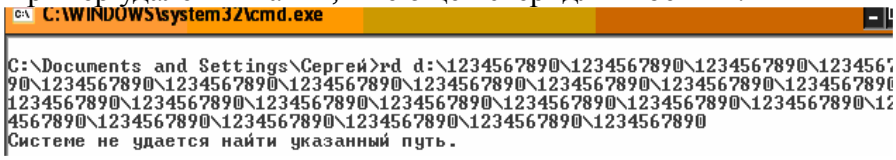
Анализатор длины файлов в папке.

В Windows длина пути к файлу допускается до 248 символов, Полная длина имени файла – до 260 символов.

Каталог со сверхдлинным именем - это каталог, у которого длина пути к нему вместе с именем самого каталога превышает 255 символов (Windows XP/7) или 127 (Windows 95/98/ME).

При наличии таких имён у операционной системы возникают проблемы с удалением, переименованием и копированием данного каталога.

Пример удаления папки, имеющей сверхдлинное имя:



На экран выводится сообщение о том, что системе не удается найти указанный путь.

Для работы с такими файлами нужно найти способ сократить длину имени.

1. Сократить длину имени можно с помощью команды “SUBST”. Эта команда связывает маршрут (путь) с буквенной меткой диска. Метка диска может быть выбрана значительно короче имени файла или папки. Присвоенная буквенная метка представляет псевдодиск, который можно использовать как обычный физический диск. Синтаксис команды выглядит следующим образом:

SUBST [диск1: [диск2:]маршрут] – создание псевдодиска
SUBST диск1: /D – удаление псевдодиска.

Для определения того, какие псевдодисков уже есть в системе используется команда SUBST без параметров.

Например, следующая команда создает псевдодиск E: для маршрута D:\HSE1\HSE2\HSE3:

>subst e: d:\HSE1\HSE2\HSE3.

При создании псевдодиска с «путем» к нужной папке, можно значительно, сократить длину «пути».

2. Сократить длину имени можно за счёт архивации информации. Для этого необходимо наблюдать за изменением длины имени при переходе на более глубокие слои операционной системы, найти такое имя, которое ещё не воспринимается ОС как сверхдлинное имя и запустить архиватор для создания архива из слишком удалённого от корня диска массива информации. Практика показывает, что всё, что находится в папках

и файлах с длиной имени более 200 символов должно быть перенесено в архив. Получится, что длина имени архива незначительно превысит 200 символов и не достигнет предельно допустимой величины.

Проверка папок на наличие длинных имён.

Программа superlong.cs

```
//Проверка папок на наличие длинных имён

using System;

using System.IO;


namespace DirSize

{

public class ShowDirSize

{

        public static long DirSize(DirectoryInfo d)

        {

         long Size=0;

         FileInfo[] fis=d.GetFiles();

          foreach (FileInfo fi in fis)

          {

           Size += fi.Length;

          if (fi.FullName.Length > 220)

         Console.WriteLine("fi  =   "  + fi + "   " + fi.FullName.Length);

          }


         DirectoryInfo[] dis = d.GetDirectories();

           foreach (DirectoryInfo di in dis)

           {

             Size += DirSize(di);

             if (di.FullName.Length > 200)
```

```
   Console.WriteLine("di = " + "  " + di + "   " + di.FullName.Length);
              }
             return (Size);
            }
            public static void Main(string[] args)
            {
             DirectoryInfo d = new DirectoryInfo(@"d:\2016н\");
             Console.WriteLine("Размер {0} поддиректориев составят {1} байтов."
   , d, DirSize(d));
            }
        }
   }
```

Красным цветом выделен адрес проверяемой папки (на дискеD: папка 2016н). Компиляция файла:

D:\1>csc superlong.cs

Microsoft (R) Visual C# 2010 Compiler version 4.0.30319.1

Copyright (C) Microsoft Corporation. All rights reserved.

Исполнение файла (при выводе каждой строки указывается имя проверяемой папки, а после знака равенства в конце строки указывается длина адреса в байтах):

D:\1>superlong //Запуск программы superlong. Имя папки указано красным цветом в тексте программы.

fi = instrumentlib_lock_file 241

di = instrumentblks 217

di = instrumentblks 202

fi = nesl_utility_lock_file 229

di = ne_sli 206

fi = powerlib_lock_file 221

di = powersys 202

fi = simscape_lock_file 221

di = simscape 202

fi = Кириченко_& 2014_Администрирование компьютерных сетей.pdf 236

fi = Библиотека для реализации генетических алгоритмов в .NET Framework .doc 253

• • •

fi = отчет Измайлов Рамиль.docx 258

fi = ОТЧЕТМ~1.DOC 244

fi = Отчет_Левшенкова.docx 253

di = Результат КР 1 231

di = 221112~1 216

di = 221112 Занятие 4 Brain Maker - и Технология нейросетевого исследования 207

fi = Лекция 3 Подготовка данных для нейронной сети.ppt 237

//---------- конец проверки

Программа зафиксировала максимальную длину имени 258 символов.
Для сравнения ниже выведена часть дерева файлов этой папки с записью его в файл superlong.doc. Файл занимает 138 страниц.

```
Структура папок тома 1
Серийный номер тома: 93B7-6274
D:\2016Н
├───Турбо С и Борланд
│   ├───13_TurboC20
│   └───ПЗ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ СИ
│       └───ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ СИ
│           ├───Турбо С и Борланд
│           └───169
│               └───pictures
├───Поиск NEW литературы
│   ├───Sharky data
│   └───Школа Матлаб
├───Нейронные сети, генетические
│   ├───helper
│   └───galib-2-4-7
│       ├───ga
│       ├───examples
│       │   ├───results
│       │   ├───pvmpop
│       │   ├───pvmind
│       │   ├───graphic
│       │   │   └───bitmaps
│       │   └───gnu
│       └───doc
│           └───images
│
```

├──Результаты 2013k факультатив Нейронные сети

...

Технология комплексирования программных средств.

Технология комплексирования программных средств предусматривает использование различных нетрадиционных действий, включая:

1. использование системы команд операционной системы
2. CMD-комплексирование программных средств
3. Переадресацию потоков.
4. Конвейер команд.
5. Исследование возможностей программы ОС «Find.exe»
6. Возможности программы ipconfig.
7. Нетрадиционное использование Интернет (использование почтовых роботов (шлюзов) для поиска и получения информации)
8. использование недокументированных возможностей C# (использование команд и программ ОС, устранение сверхдлинных имён (superlong), и др.).
9. Вызов команд ОС из C#-программы
10. Использование команд и готовых программ ОС из C#-программы
11. установка среды разработки, средств контроля и настройка преобразователей
12. Встраивание в программу внешних пакетов
13. Нейронные вставки в создаваемую программу

Команды и управляющие программы ОС.

Система команд ОС содержит команды, среди которых можно выделить команды справочной системы ОС, команды файловой системы, команды управления работой ОС, команды пакетных файлов и сетевые команды.

Существует три способа использования этих команд:

1. Разрозненное использование команд;
2. Конвейеризация команд ОС;
3. Группировка команд в программы (скрипты).

Разрозненное использование системы команд ОС требует наличия командного процессора. В ОС Windows используется два командных процессора: command.com и cmd.exe. Первый из них – это 16-битный, доставшийся в наследство от операционной системы DOS. Второй – 32-битный, для более поздних операционных систем.

Оба командных процессора используют практически одинаковый состав команд. Но cmd.exe, как более поздний, имеет дополнительные возможности, обеспечиваемые микропроцессорами, начиная с i486. К ним относятся: система защиты памяти, возможность работы с длинными именами, и др.

Конвейеризация команд операционной системы позволит соединить команды ОС для совместного использования. В этом случае благодаря использованию специальной команды создания конвейера (|) результат работы одной команды ОС (т.е. выходной поток этой команды) передаётся другой команде в виде её входного потока.

Группировка команд ОС в программы позволяет создавать командные файлы (скрипты). Для командного процессора command.com командные файлы имеют расширение .bat. Для процессора cmd.exe командные файлы имеют расширение .cmd.

Необходимо отметить, что каждый командный процессор оперирует двумя типами команд – внутренними и внешними. Внешние команды реализуются с помощью исполняемых файлов, размещённых в папке System32 операционной системы. Поскольку состав таких файлов в этой папке может быть изменён, допускается расширение состава внешних команд по сравнению со стандартным, поставляемым вместе с ОС.

Некоторые пакеты прикладных программ содержат несколько вариантов исполнения – для использования графического режима, и для командного режима. Например, архиватор RAR имеет две разновидности основной программы. Причём, программа, предназначенная для командного режима работы, имеет больше возможностей. В состав пакета включён специальный файл с описанием команд архиватора.

Пакеты, допускающие командный режим управления, могут быть использованы в командных файлах ОС и фактически являются расширением системы команд ОС. Командный режим управления позволяет легко объединять программы, создавая функционально различающиеся программные системы.

В значительной мере возможности ОС расширяются за счёт использования программ, допускающих командное управление. Кроме упомянутого архиватора RAR – внешней по отношению к ОС программе – аналогичные возможности предоставляют программы ОС msinfo32, findstr, sort, ftype, xcopy, debug, ntsd и др. Так например, программа debug позволяет выполнять запись сектора на магнитном носителе, использовать программные вставки в виде команд ассемблера, производить чтение и запись основной памяти и регистров микропроцессорного комплекта ЭВМ, и т.д.

Очень большое значение для комплексирования программных средств имеет возможность использовать команды ОС из языков высокого уровня.

В отличие от C++ в языке C# не предусмотрено использование в программе ассемблерных вставок. Может сложиться впечатление, что использование ассемблерных вставок для C# недоступно. Это обманчивое впечатление. Команды псевдоассемблера можно использовать в C#, обратившись к стандартным средствам ОС – программам debug и ntsd. Кроме того, такие пакеты, как MASM разрешают режим командного управления и делают работу с ассемблером доступной для C#.

В связи со всем изложенным, процесс исследования возможности повышения производительности программиста, работающего в рыночных условиях за счёт комплексирования программных средств, построим следующим образом:

1. Анализ системы команд ОС Windows
2. Исследование возможностей практического использования системы команд ОС
3. Определение возможности использования системы команд ОС из языка высокого уровня.

Система команд Windows.

Система команд Windows включает в себя:

- Команды справочной системы
- Команды файловой системы
- Команды управления работой ОС
- Команды пакетных файлов

1. Команды справочной системы

HELP	Выводит справочную информацию о системе команд Windows_2000/XP.
HELP имя_команды	Выводит справочную информацию для

	набранной команды
Имя_команды /?	Выводит справочную информацию для набранной команды

2. Команды файловой системы

ATTRIB	Отображение и изменение атрибутов файлов.
CD	Смена текущей папки.
CHDIR	Вывод имени либо смена текущей папки.
CHKDSK	Проверка диска и вывод статистики.
CLS	Очистка экрана.
COMP	Сравнение содержимого двух файлов или двух наборов файлов.
COPY	Копирование одного или нескольких файлов в другое место.
DEL	Удаление одного или нескольких файлов.
DIR	Вывод списка файлов и подпапок из указанной папки.
DISKCOMP	Сравнение содержимого двух гибких дисков
DISKCOPY	Копирование содержимого одного гибкого диска на другой.
ERASE	Удаление одного или нескольких файлов.
FC	Сравнение двух файлов или двух наборов файлов и вывод различий между ними.
FIND	Поиск текстовой строки в одном или нескольких файлах.
FINDSTR	Поиск строк в файлах.
FORMAT	Форматирование диска для работы с Windows 2000/XP.
LABEL	Создание, изменение и удаление меток тома для дисков.
MD	Создание папки.
MKDIR	Создание папки.
MOVE	Перемещение одного или нескольких файлов из одной папки в другую.
PUSHD	Сохранение значения текущей активной папки и переход к другой

	папке.
POPD	Восстановление предыдущего значения текущей активной папки, сохраненного с помощью команды PUSHD.
PRINT	Вывод на печать содержимого текстовых файлов.
RD	Удаление папки.
REN	Переименование файлов и папок.
RENAME	Переименование файлов и папок.
REPLACE	Замещение файлов.
RMDIR	Удаление папки.
SORT	Сортировка ввода.
TREE	Графическое отображение структуры папок заданного диска или заданной папки.
TYPE	Вывод на экран содержимого текстовых файлов.
VERIFY	Установка режима проверки правильности записи файлов на диск
VOL	Вывод метки и серийного номера тома для диска.
XCOPY	Копирование файлов и дерева папок.

3. Команды управления работой ОС

ASSOC	Вывод либо изменение сопоставлений по расширениям имен файлов.
AT	Выполнение команд и запуск программ по расписанию.
BREAK	Включение/выключение режима обработки комбинации клавиш CTRL+C.
CACLS	Отображение/редактирование списков управления доступом (ACL) к файлам.
CHCP	Вывод либо установка активной кодовой страницы.
CHKNTFS	Отображение или изменение выполнения проверки диска во время загрузки.
CMD	Запуск еще одного интерпретатора командных строк Windows

	2000/XP.
COLOR	Установка цвета текста и фона, используемых по умолчанию.
COMPACT	Отображение/изменение сжатия файлов в разделах NTFS.
CONVERT	Преобразование дисковых томов FAT в NTFS.
DATE	Вывод либо установка текущей даты.
DOSKEY	Редактирование и повторный вызов командных строк; создание макросов.
FTYPE	Вывод либо изменение типов файлов, используемых при сопоставлении по расширениям имен файлов.
GRAFTABL	Позволяет Windows 2000/XP отображать расширенный набор символов в графическом режиме.
MODE	Конфигурирование системных устройств.
MORE	Последовательный вывод данных по частям размером в один экран.
PATH	Вывод либо установка пути поиска исполняемых файлов.
PROMPT	Изменение приглашения в командной строке Windows 2000/XP.
RECOVER	Восстановление читаемой информации с плохого или поврежденного диска.
SET	Вывод, установка и удаление переменных среды Windows 2000/XP.
START	Запуск программы или команды в отдельном окне.
SUBST	Сопоставляет заданному пути имя диска.
VER	Вывод сведений о версии Windows 2000/XP.
| (вертикальная черта)	Перенаправление выходного потока на вход следующей команды (реализация конвейера команд)
>	Переопределение выходного потока (например, направление вывода в начало текстового файла)
>>	Перенаправление выходного потока (например, направление вывода в конец текстового файла)
<	Перенаправление входного потока

	(например, ввод команд не с клавиатуры, а из текстового файла)

4. Команды пакетных файлов

CALL	Вызов одного пакетного файла из другого.
ECHO	Вывод сообщений и переключение режима отображения команд на экране.
ENDLOCAL	Конец локальных изменений среды для пакетного файла.
EXIT	Завершение работы программы CMD.EXE (интерпретатора командных строк).
FOR	Запуск указанной команды для каждого из файлов в наборе.
GOTO	Передача управления в отмеченную строку пакетного файла.
IF	Оператор условного выполнения команд в пакетном файле.
PAUSE	Приостановка выполнения пакетного файла и вывод сообщения.
REM	Помещение комментариев в пакетные файлы и файл CONFIG.SYS.
SETLOCAL	Начало локальных изменений среды для пакетного файла.
SHIFT	Изменение содержимого (сдвиг) подставляемых параметров для пакетного файла.

Конвейеризация команд Windows.

Конвейер команд

Выделять из перехваченных данных необходимые для дальнейшего использования можно, используя конвейер команд.

Рассмотрим выдачу, полученную командой dir для корневого директория диска C:\.

```
28.03.2006  23:31    <DIR>          777
```

```
25.01.2007  19:16    <DIR>          Из HSE
17.09.2005  15:56    <DIR>          2
10.09.2005  16:05    <DIR>          1
21.11.2005  22:47    <DIR>          PerfLogs
04.12.2005  19:24    <DIR>          temp
15.01.2006  16:57    <DIR>          2006н
02.03.2006  18:12    <DIR>          Magic Waterfall Screensaver
04.03.2006  13:52    <DIR>          Test_System
02.05.2006  16:22    <DIR>          WebServers
02.02.2004  22:26    <DIR>          DVD
17.05.2006  15:54           410 543 braun_dyen_kod_da_vinchi.rtf.zip
11.05.2006  16:19           184 320 Olympus 310.doc
06.10.2006  11:12            26 624 hakers word.doc
03.01.2005  16:44    <DIR>          1с
25.03.2004  11:09    <DIR>          ACTIVstudio
21.02.2005  18:30               182 my_first.pl
18.07.2005  13:07    <DIR>          2005-2006 уч год
11.08.2005  11:48    <DIR>          JPG-BMP
               9 файлов        622 331 байт
              29 папок     254 394 368 байт свободно

C:\>
```

Надо отметить, что этот директорий очень напряжённый и насыщенный. Чаще всего он по команде dir не умещается в окне консоли команд.

Здесь приведен лишь фрагмент вывода по команде dir. Обратим внимание, что кроме огромного перечня файлов и папок команда dir даёт и сводные результаты:

в директории хранится 9 файлов

общим объёмом 622 331 байт, и

29 папок.

При этом на диске, в корневой директорий которого мы заглядывали, свободно 254 394 368 байт.

Это значит, что команда dir даёт достаточное количество информации, чтобы ответить на такие вопросы:

- Сколько файлов содержится в данном каталоге?
- Какой объём занимают файлы, хранящиеся в данном каталоге?
- Сколько папок содержится в данном каталоге?
- Сколько свободного места есть на данном диске?

Но чтобы получить ответы на эти вопросы, надо переворошить груду информации и найти в ней то, что Вас интересует.

Ответ должен быть коротким, понятным и информативным.

Чтобы удовлетворить этим требованиям, одной команды dir недостаточно.

Возможности этой (и других команд) можно расширить с помощью конвейера.

Вспомним, что есть ещё такая команда find, по которой можно среди большого количества строк найти ту, которая больше всего интересует.

Конвейер команд позволяет выдачу, полученную по одной команде, предоставить для анализа другой команде.

Построим конвейер из двух команд: dir и find.

Первая команда получит информацию о корневом директории диска C:, а вторая позволит выделить из этой информации (иногда – очень большого объёма) нужную.

Такое сочетание команд в конвейере позволяет ответить на вопрос:

«Сколько файлов хранится в данной директории и каков их объём?».

Выполним в консоли команд:

```
C:\>dir | find "файлов"
              19 файлов        136 257 байт
```

Здесь для команды find в кавычках мы указали поисковое предписание: что искать?

А если вместо слова «файлов» указать слово «байт»?

```
C:\>dir | find "байт"
               19 файлов        136 257 байт
```

Получили то же самое.

Внутри одной строки программа find неспособна разделить информацию.

Использование конвейера команд.

Сколько файлов хранится в данной директории и каков их объём?

Для ответа на вопрос, заданный в заголовке, составим запрос:

```
C:\77>dir | find "файлов"
              19 файлов        136 257 байт
```

Для этой цели можно воспользоваться и другой программой: findstr.

```
C:\>dir AX1 | findstr  "файлов"
              25 файлов      3 444 924 байт
```

2.2. Какие текстовые файлы хранятся в этом директории?

Для ответа на этот вопрос составим запрос:

```
C:\77>dir | find ".txt"
```

и получим ответ:

```
05.11.2006  12:36             23 004 test0-1.txt
04.11.2006  17:38           16 572 test0-1v2.txt
```

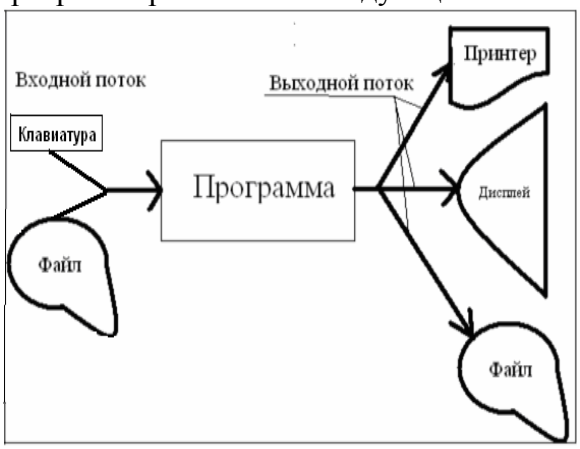
В директории найдено два файла с расширением txt.

Практическое использование системы команд Windows.

Перехват потока.

«Перехватить поток» - это значит перенаправить его туда, где он необходим.

Практически любая программа работает по следующей схеме:



Основным входным потоком является поток с клавиатуры. Основным выходным потоком является вывод на дисплей.

Перенаправление потока в операционной системе Windows выполняется с помощью трёх команд:

<

>

>>

Команда “<” - это команда перенаправления **входного** потока.

Острие команды показывает, куда она направлена.

Например: **команда**

debug.exe < file.txt

направляет входной поток из файла file.txt на вход программы debug.exe.

Это значит, что в файле file.txt записана последовательность команд, которую должна выполнить программа debug.exe.

Команда “>” это команда перенаправления выходного потока.

Острие команды так же показывает, куда она направлена.

Например:

Debug.exe > file1.txt.

означает, что результаты работы программы debug.exe будут направляться не на экран (как обычно), а в файл file1.txt.

Если этого файла не существует, он будет создан. А если он существует, его содержимое будет заменено результатами работы программы debug.exe. Т.е. информация в этот файл поступит, начиная с самого его начала.

Команда “>>” это команда перенаправления выходного потока. Острие команды так же показывает, куда она направлена.

Например:

Debug.exe >> file2.txt.

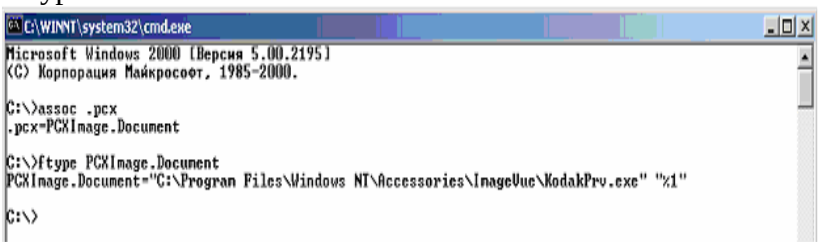
означает, что результаты работы программы debug.exe будут направляться не на экран, а в файл file2.txt.

Если этого файла не существует, он будет создан. А если он существует, его содержимое будет дополнено результатами работы программы debug.exe. Причём эти результаты ничего не уничтожат в файле, они дополнят старое содержимое файла (новые данные будут дописаны в его конец).

Как узнать адрес программы, имеющейся в ОС Windows?

Тип файла характеризуется его расширением. Для обработки файлов определённого типа предназначена своя программа.

- Определить тип файла можно с помощью команды assoc.
- Имя обрабатывающей файлы этого типа программы и её адрес можно получить командой ftype:



Пример1:

Определить, какая программа обрабатывает файлы с расширением .pcx?

По команде assoc получаем:

```
.pcx=PCXImage.Document
```

В этом ответе находим, что файлы pcx относятся к типу «PCXImage.Document».

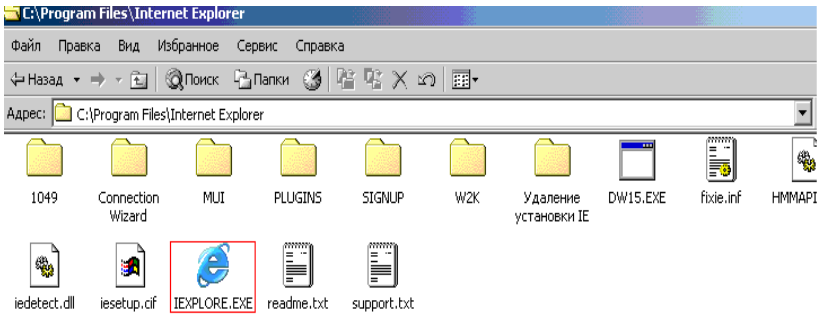
Определим, кто является обработчиком этого типа файлов и где находится сам обработчик:

```
C:\>ftype PCXImage.Document
PCXImage.Document="C:\Program Files\Windows
NT\Accessories\ImageVue\KodakPrv.exe" "%1"
```

Здесь жирным шрифтом выделен путь к обработчику, курсивом – сам обработчик. После имени обработчика указывается передаваемый обработчику параметр.

Пример 2: надо узнать, где находится обработчик файлов, имеющих расширение htm. Выполняем:

```
C:\>assoc .htm
.htm=htmlfile
C:\>ftype htmlfile
htmlfile="C:\Program Files\Internet Explorer\iexplore.exe"
```



Создание командного файла, использующего программу Find.exe.

Создадим командный файл Fstring.bat, использующий программу Find.exe.

```
C:\>copy con fstring.bat
find /%1 "%2" c:/%3
^Z
Скопировано файлов:        1.
```

В данном примере:

%1 – переменная, запрашивающая параметр вывода ([/V] [/C] [/N] [/I]).

%2 – переменная, запрашивающая строку для поиска.

%3 – переменная, запрашивающая файл с диска “C:” в котором будет искаться данная строка.

Далее создадим текстовый файл в котором мы будем искать строку.

```
I asked him about comand files
And what do you think he answered
He told that it is very simple
To make it properly
```

Теперь введем команду fstring 1 2 3 , где 1, 2 и 3 – параметры функции.
Например:

```
C:\>fstring n do sample.txt

C:\>find /n "do" c:/sample.txt

---------- C:/SAMPLE.TXT
[2]And what do you think he answered
```

Управление из командной строки программой msinfo32.

1. Помимо средства **«Сведения о системе»** для просмотра данных об управляемом компьютере можно использовать команду **msinfo32**.

Msinfo32 служит для отображения подробных сведений об оборудовании, системных компонентах и среде программного обеспечения.

Синтаксис

msinfo32 [**/?**] [**/pch**] [**/nfo** *имя_файла*] [**/report** *имя_файла*] [**/computer** *имя_компьютера*] [**/showcategories**] [**/category** *код_категории*] [**/categories** *код_категории*]

Параметры

имя_файла - файл, который требуется открыть. Файл может иметь расширение NFO, XML, TXT или CAB.

/? - отображение справки по команде msinfo32.

/pch - отображение журнала.

/nfo *имя_файла* - сохранение экспортированного файла как NFO-файла.

/report *имя_файла* - сохранение экспортированного файла как TXT-файла.

/computer *имя_компьютера* - открытие окна сведений о системе для указанного удаленного компьютера.

/showcategories - открытие окна сведений о системе, содержащего все доступные коды

категорий.
/category *код_категории* - Открытие окна сведений о системе, в котором выбрана указанная категория. Для отображения списка доступных кодов категорий служит параметр **/showcategories**.
/categories *код_категории* - открытие окна сведений о системе, содержащего только указанные категории. Вывод также ограничивается только выбранными категориями. Для отображения списка доступных кодов категорий служит параметр **/showcategories**

Примеры

Чтобы получить список доступных кодов категорий, введите:

```
msinfo32 /showcategories
```

Чтобы открыть окно сведений о системе, содержащее все доступные сведения, кроме сведений о загруженных модулях, введите:

```
msinfo32 /categories +all -loadedmodules
```

Чтобы открыть окно сведений о системе и создать NFO-файл syssum.nfo, содержащий сведения категории «Сведения о системе», введите:

```
msinfo32 /nfo syssum.nfo /categories +systemsummary
```

Чтобы вывести сведения о конфликте ресурсов и создать NFO-файл conflicts.nfo, содержащий сведения о конфликтах ресурсов, введите:

```
msinfo32 /nfo conflicts.nfo /categories
```

+componentsproblemdevices+resourcesconflicts+resourcesforcedhardware

Команда FINDSTR.

Findstr - **поиск образцов текста в файлах с использованием регулярных выражений.**

Формат комнады: findstr [**/b**] [**/e**] [**/l**] [**/r**] [**/s**] [**/i**] [**/x**] [**/v**] [**/n**] [**/m**] [**/o**] [**/p**] [**/offline**] [**/g:***файл*] [**/f:***файл*] [**/c:***строка*] [**/d:***СписокКаталогов*] [**/a:***АтрибутЦвета*] [*строки*] [[*диск***:**][*путь*] *ИмяФайла* [...]], где

строки - текст, поиск которого производится в файле, заданном параметром *ИмяФайла*.

[*диск***:**][*путь*] *ИмяФайла* [...] - файл или несколько файлов для поиска

Ключи команды:

/b
: Сравнивает шаблон с началом строки.

/e
: Сравнивает шаблон с концом строки.

/l
: Использует заданную строку буквально.

/r
: Использует строку поиска как регулярное выражение. Команда Findstr интерпретирует все метасимволы как регулярные выражения, если не используется ключ **/l**.

/s
: Задает поиск файлов в текущем каталоге и его подкаталогах.

/i
: Задает поиск без различия строчных и заглавных букв.

/x
: Печатает точно совпавшие строки.

/v
: Печатает строки, не содержащие совпадений.

/n
: Печатает в начале совпавшей строки ее номер.

/m
: Печатает только имя файла при обнаружении совпадения.

/o

Печатает смещение перед выводом строки с совпадением.

/p

Пропускает файлы с непечатаемыми символами.

/offline

Обработка файлов с автономным атрибутом.

/f:*файл*

Читает список из заданного файла.

/c:*строка*

Использует заданный текст как литеральную строку поиска.

/g:*файл*

Получает строки поиска из заданного файла.

/d:*СписокКаталогов*

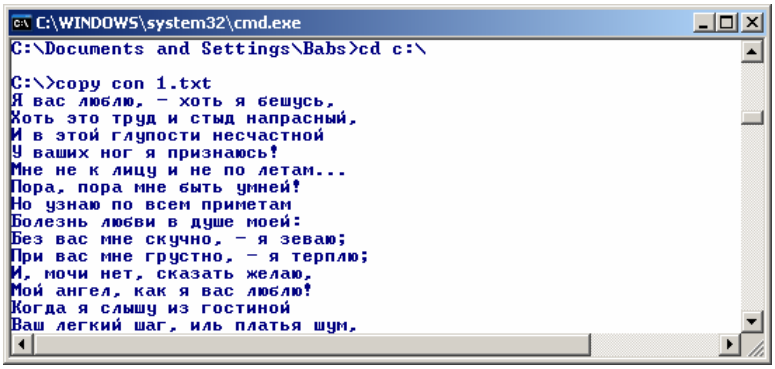
Ищет в списке каталогов, разделенном запятыми.

/a:*АтрибутЦвета*

Задает атрибуты цвета двумя шестнадцатеричными цифрами.

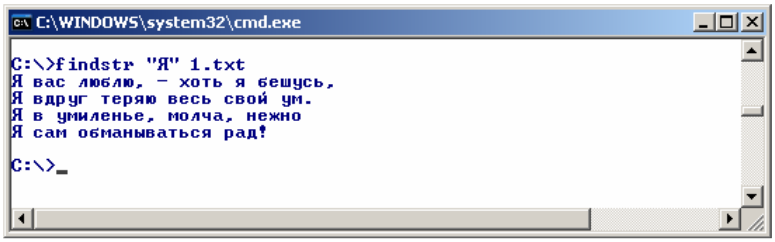
Пример использования команды findstr:

Сначала создадим текстовый файл 1.txt, используя команду copy con:

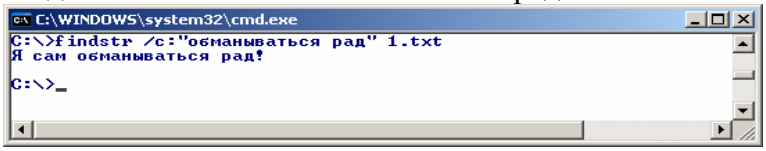


Теперь найдем строки содержащие слово «Я». Для этого введем команду:

C:\>findstr "Я" 1.txt



Если нам необходимо найти целую строку или определенное словосочетание, то воспользуемся командой: C:\>findstr /c:"обманываться рад" 1.txt



Для вывода на экран строк не содержащих совпадений воспользуемся командой:

C:\>findstr /v /c:"обманываться рад" 1.txt

Команда - FTYPE

Просмотр и изменение типов файлов, сопоставленных с расширением имен файлов

FTYPE [типФайлов[=[командная СтрокаОткрытия]]]

типФайлов - Тип файлов для просмотра или изменения

командная СтрокаОткрытия - Команда, используемая для открытия файлов

указанного типа.

- Команда FTYPE без параметров выводит текущий список типов файлов, для которых определены командные строки открытия.
- Если указан только тип файла, FTYPE выводит командную строку открытия для этого типа файлов.
- Если после знака равенства не указана строка открытия, FTYPE удалит текущее сопоставление для указанного типа файлов.
- При вызове командной строки переменные %0 и %1 заменяются на имя файла, запускаемого с помощью сопоставления.
- Вместо переменной %* подставляются все параметры, а переменные %2, %3 и т.д. заменяются, соответственно, на первый, второй и другие параметры.
- Вместо переменной %~n подставляются все оставшиеся параметры, начиная с n, где n является числом от 2 до 9. Например:

ASSOC .pl=PerlScript
FTYPE PerlScript=perl.exe %1 %*

Эти команды позволят вызывать обработчик команд Perl следующим образом:

script.pl 1 2 3

- Если желательно избежать постоянного ввода расширения имен файлов, введите следующую команду:

set PATHEXT=.pl;%PATHEXT%

Теперь обработчик команд вызывается еще проще:

script 1 2 3

Команда subst.

Сопоставление имени диска указанному пути.

Формат команды:

SUBST [диск1: [диск2:]путь]
SUBST диск1: /D

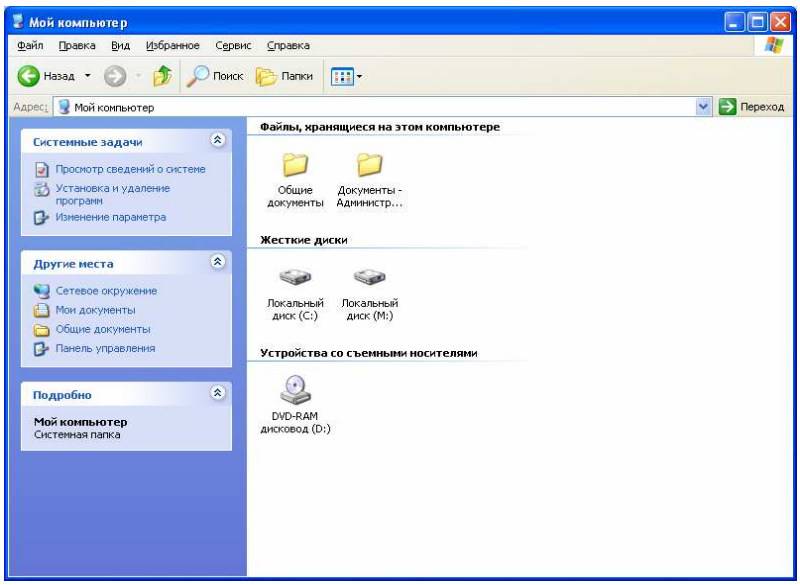
где

диск1: Виртуальный диск, который сопоставляется указанному пути.
[диск:]путь Физические диск и путь,
которым сопоставляется виртуальный диск.
/D Удаление ранее созданного виртуального диска.

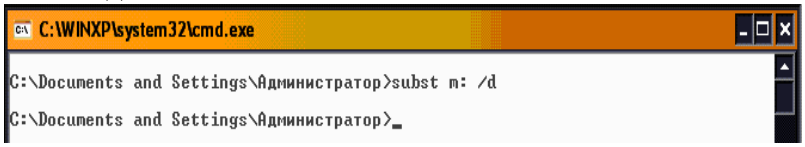
SUBST без параметров - вывод текущего списка виртуальных дисков.

Создадим виртуальный диск M, который сопоставляется с путём C:\newdisk:

Таким образом, в папке «Мой компьютер» появится новый диск M:



Теперь удалим этот диск:



Соответственно, он перестанет существовать и в «Моём компьютере»:

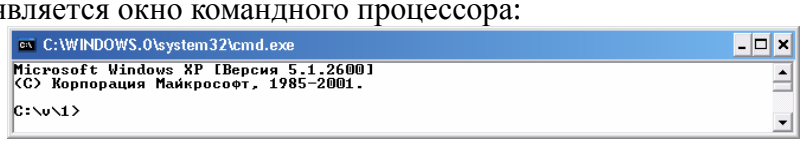
Использование системы команд ОС из C#-программы.

Запуск командного процессора

Для запуска 32-разрядного командного процессора операционной системы создаём процесс с идентификатором my, в свойстве FileName указываем полный адрес командного процессора и запускаем процесс:

```
using System;
using System.Diagnostics;

class MyProcess
{
public static void Main()
{
Process my = new Process();
my.StartInfo.FileName = "C:\\WINDOWS.0\\system32\\cmd.exe";
my.Start();
Console.Read();
}
}
```

В результате появляется окно командного процессора:



Ресурсы процесса и передача параметров запускаемому процессу

В операционной системе Windows запустить процесс можно двумя способами:

1) Запустить обрабатывающую программу и с помощью параметров передать этой программе имя обрабатываемого документа;

2) Дважды щёлкнуть по файлу, содержащему обрабатываемый документ. Операционная система по расширению файла определит, какая программа обрабатывает документ такого типа, вызовет на исполнение эту программу и передаст ей

обрабатываемый документ.

Вызов программы: указанием ресурса.

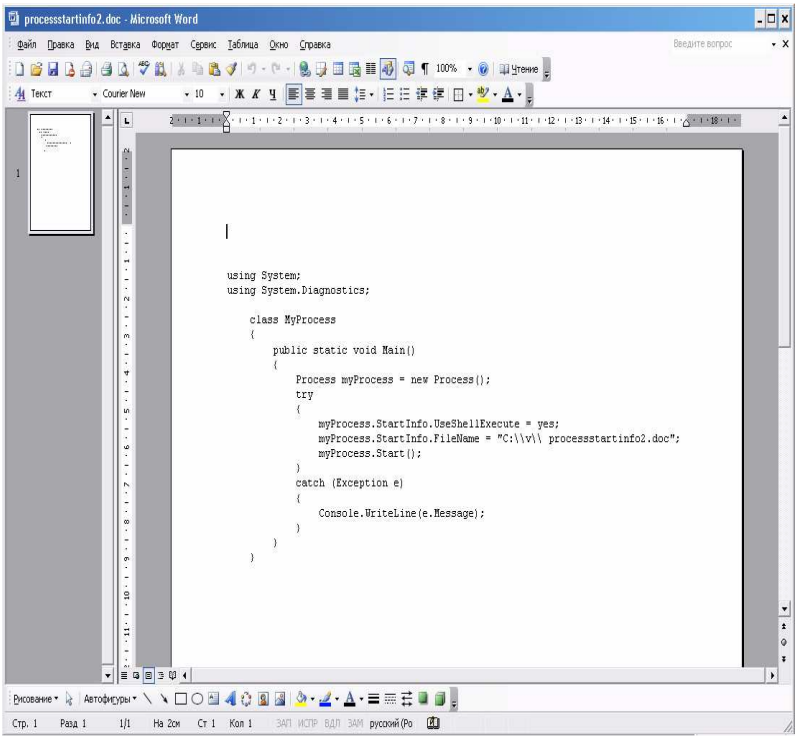
C#-программа processstartinfo2.exe используется для запуска файла, содержащего исходный код программы. Запуск процесса производится указанием имени запускаемого ресурса. Поскольку имя программы для обработки данного ресурса не указывается, необходимо разрешить операционной системе использовать свои возможности. Для этого свойство StartInfo.UseShellExecute получает значение true:

```
using System;
using System.Diagnostics;

    class MyProcess
    {
        public static void Main()
        {
            Process myProcess = new Process();
            try
            {
                myProcess.StartInfo.UseShellExecute = true;
            myProcess.StartInfo.FileName = "C:\\v\\processstartinfo2.doc";
                myProcess.Start();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
```

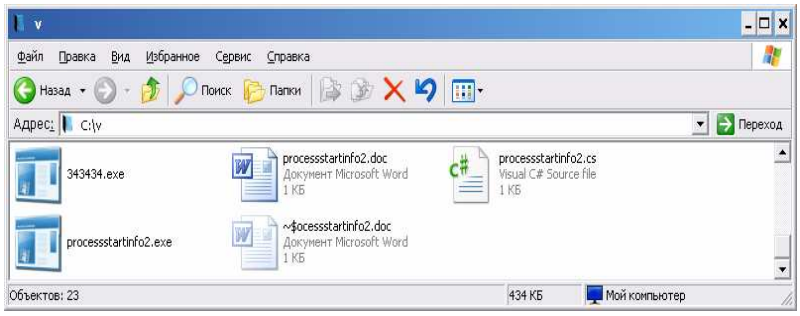
Все необходимые файлы находятся в папке v. Протокол запуска программы:

```
C:\v>processstartinfo2

C:\v>
```

На экране появляется окно Word, содержащее исходный код программы processstartinfo2.cs:



Фрагмент окна папки v:



В данном случае C#-программа была запущена двойным щелчком по ресурсу (другими словами – по документу, который должен быть открыт).

Перехват результатов работы процесса

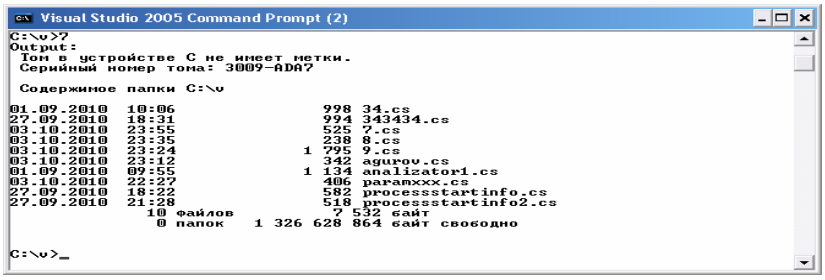
Процесс создаётся для того, чтобы выполнить какую-то часть работы, а затем использовать полученный результат. С этой целью выполним программу 7.cs:

```
//Программа 7.cs
using System;
using System.Diagnostics;

class MainClass
{
    public static void Main()
    {
        Process p = new Process();
        p.StartInfo.FileName = "cmd.exe";
        p.StartInfo.Arguments = "/c dir *.cs";
        p.StartInfo.UseShellExecute = false;
        p.StartInfo.RedirectStandardOutput = true;
        p.Start();

        string output = p.StandardOutput.ReadToEnd();

        Console.WriteLine("Output:");
        Console.WriteLine(output);    }
}
```

В этой программе с помощью 32-разрядного процессора cmd.exe запрашивается, какие исходные тексты программ, написанных на языке c# содержатся в текущем директории. Имеется в виду директорий v: в корне диска C:.

В свойстве StartInfo.Arguments содержится комбинация символов /c, обозначающая необходимость выполнения команды операционной системы – далее указывается, что такой командой должна быть команда dir. Результат выполнения этой команды разрешено перенаправить в стандартный выходной поток.

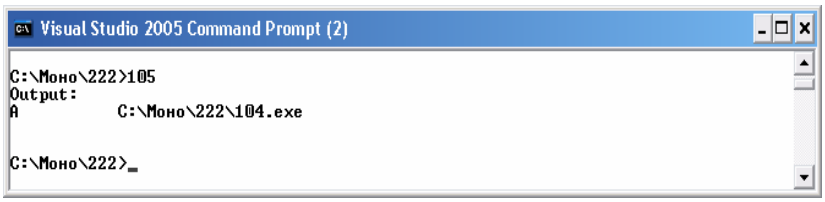
Последние три команды посвящены работе с этим потоком. Сначала стандартный выходной поток направляется в переменную output. Затем выводится на печать заголовок, предупреждающий, что будет выводиться содержимое стандартного выходного потока, т.е. результат работы команды dir операционной системы. И затем распечатывается содержание переменной output, т.е. список имеющихся в каталоге v исходных текстов c#-программ.



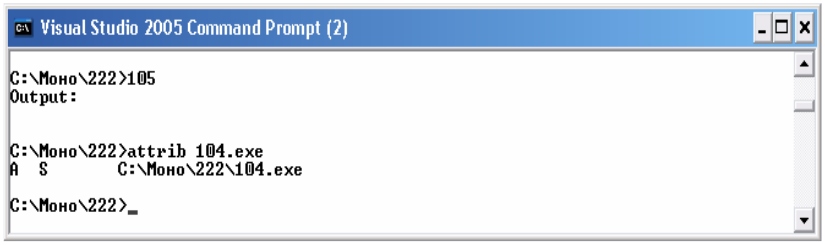
Установка атрибутов файла

Получить информацию об атрибутах файла с помощью команд ОС можно, указав в свойстве StartInfo.Arguments команду attrib и имя файла:

```
using System;
using System.Diagnostics;

class MainClass
{
    public static void Main()
    {
        Process p = new Process();
        p.StartInfo.FileName = "cmd.exe";
        p.StartInfo.Arguments = "/c attrib 104.exe";
        p.StartInfo.UseShellExecute = false;
        p.StartInfo.RedirectStandardOutput = true;
        p.Start();

        string output = p.StandardOutput.ReadToEnd();

        Console.WriteLine("Output:");
        Console.WriteLine(output);    }
}
```



Полученная информация говорит о том, что файл 104.exe имеет атрибут A (архивный). Если изменить свойство StartInfo.Arguments на "/c **attrib +S** 104.exe", присвоим файлу ещё и атрибут S - ‘системный’:



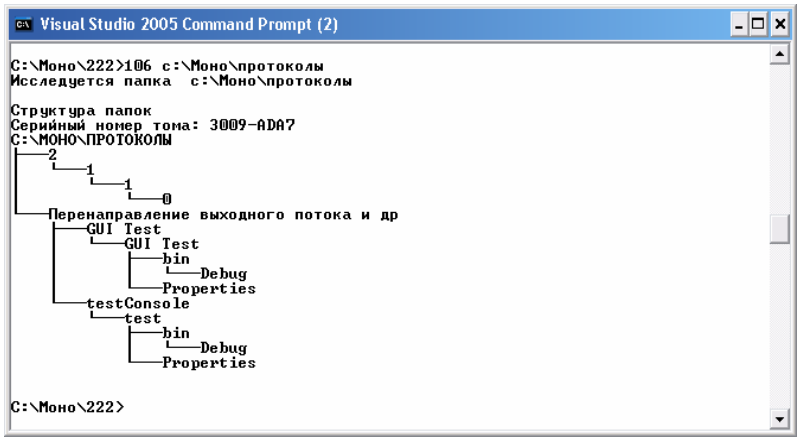
При этом сам процесс присвоения никак не отражается в листинге: программа 105.exe в выходной поток ничего не направляет.

Получение структуры папки по команде tree

Из C#-программы можно получить структуру любой папки в виде дерева с помощью команды операционной системы tree. Воспользуемся для этого программой 106.cs.

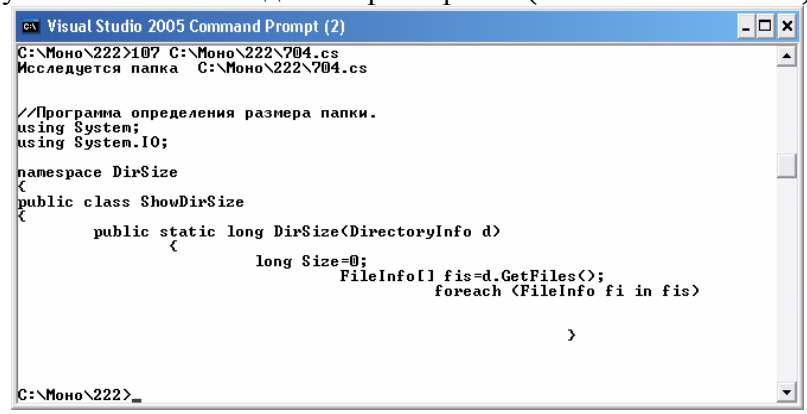
```
//Программа 106.cs: Структура папки в виде дерева.
//Имя папки указать в командной строке при запуске программы.
using System;
using System.Diagnostics;
using System.IO;

      class Class1
      {
            static void Main(string[] args)
            {
                        string dir = args[0];
     Console.WriteLine("Исследуется папка  " + dir + "\n");
```

```
                  ProcessStartInfo startinfo;
            Process          process = null;
            string           command, stdoutline;
            StreamReader     stdoutreader;

            // Команда которую будет выполнять
            command = "tree ";

            try
            {

      startinfo = new ProcessStartInfo();
                  // Запускаем через cmd
                  startinfo.FileName = "cmd.exe";
                  // Ключ /c - выполнение команды
//Кроме команды можно вставить и необходимые параметры
 startinfo.Arguments = "/C " + command + dir;
                  // Не используем shellexecute
 startinfo.UseShellExecute = false;
 // Перенаправить вывод на обычную консоль
                  startinfo.RedirectStandardOutput = true;
                  // Стартуем
 process = Process.Start(startinfo);

                  // Получаем вывод
 stdoutreader = process.StandardOutput;
                  while((stdoutline=stdoutreader.ReadLine())!= null)
                  {
                        // Выводим
                        Console.WriteLine(stdoutline);
                  }
                  stdoutreader.Close();
                  stdoutreader = null;
            }
            catch
            {
                  throw;
            }
            finally
            {
                  if (process != null)
                  {
                        // Закрываем
                        process.Close();
                  }

                  // Освобождаем
                  process = null;
                  startinfo = null;
            }
            Console.ReadLine();
      }
 }
```



Вывод на экран текстового файла командой type

Если в программе 106.cs вместо команды tree вставить команду type, получим вывод на экран текста указанного в командной строке файла (C:\Моно\222\704.cs):



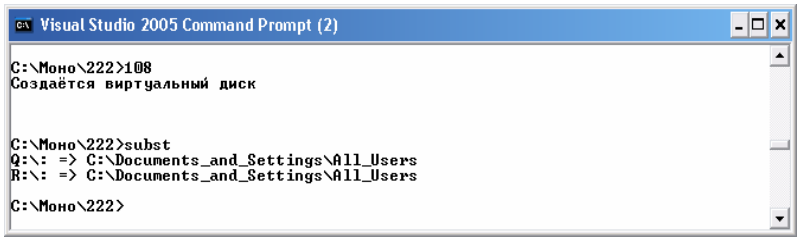
Надо только иметь в виду, что команда type выводит текст в коде ASCII. Для вывода текста в другой кодировке необходимо программу вывода перестроить.

Создание виртуального диска

Используем программу 108.cs и команду subst операционной системы для создания виртуального диска Q:.

```
//Программа 108.cs. Создание виртуального диска
using System;
using System.Diagnostics;
using System.IO;

      class Class1
      {
            static void Main(string[] args)
            {

   Console.WriteLine("Создаётся виртуальный диск " + "\n");
                        ProcessStartInfo startinfo;
                  Process          process = null;
                  string           command, stdoutline;
                  StreamReader     stdoutreader;

// Команда которую будет выполнять операционная система
                  command = "subst q: ";

                  try
                  {

            startinfo = new ProcessStartInfo();
```

```
                        // Запускаем через cmd
            startinfo.FileName = "cmd.exe";
                        // Ключ /c - выполнение команды
     //Кроме команды можно вставить и необходимые параметры
    //Задаём папку, в которой будет создан виртуальный диск
      string dir = @"C:\Documents_and_Settings\All_Users";
      startinfo.Arguments = "/C " + command + dir;
                        // Не используем shellexecute
      startinfo.UseShellExecute = false;
// Перенаправить вывод на обычную консоль
                        startinfo.RedirectStandardOutput = true;
                        // Стартуем
      process = Process.Start(startinfo);

                        // Получаем вывод
      stdoutreader = process.StandardOutput;
                        while((stdoutline=stdoutreader.ReadLine())!= null)
                        {
                              // Выводим
                              Console.WriteLine(stdoutline);
                        }
                        stdoutreader.Close();
                        stdoutreader = null;
                  }
                  catch
                  {
                        throw;
                  }
                  finally
                  {
                        if (process != null)
                        {
                              // Закрываем
                              process.Close();
                        }

                        // Освобождаем
                        process = null;
                        startinfo = null;
                  }

                  Console.ReadLine();
            }
      }
```



Проверка показывает, что в заданной папке создан виртуальный диск Q:.

Управление работой архиватора

Кроме программы WinRAR, имеющей оконный интерфейс, в комплект поставки архиватора входит файл *Rar.exe*. Это также 32-разрядная версия RAR для Windows, но она поддерживает только интерфейс командной строки и работает в текстовом (консольном) режиме. Консольную версию RAR удобно использовать для вызова из пакетных файлов (BAT и CMD), для запуска из приглашения DOS и т.п. Она поддерживает больше команд и ключей в командной строке, чем WinRAR.

Общий синтаксис командной строки для архивации файлов таков:

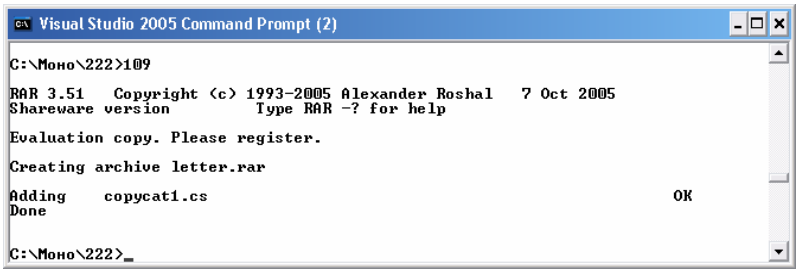
Rar.exe A [-ключи] <Архив> [Файлы] [@Файлы-списки]

Например, если необходимо добавить файл copycat1.cs в архив LETTER.RAR, используется команда:

rar.exe a letter.rar copycat1.cs

Для сохранения в архиве файла используем программу 109.cs:

```
//109.cs: Управление архиватором из C#-программы
using System;
using System.Diagnostics;
using System.IO;

      class Class1
      {
            static void Main(string[] args)
            {
                  ProcessStartInfo startinfo;
                  Process          process = null;
                  string           command;

                  // Команда которую будет выполнять
                  command = " a ";

                  try
                  {
                        string arhiv = "letter.rar";
                        string file = "copycat1.cs";
            startinfo = new ProcessStartInfo();
                        // Запускаем архиватор
                        startinfo.FileName = "rar.exe";
                        // Ключ /c - выполнение команды
    //Кроме команды можно вставить и необходимые параметры
startinfo.Arguments = "/C " + command + " " + arhiv + " " + file;
                        // Не используем shellexecute
      startinfo.UseShellExecute = false;

                        // Стартуем
      process = Process.Start(startinfo);
                  }
                  catch
                  {
                        throw;
                  }
                  finally
                  {
                        if (process != null)
                        {
                              // Закрываем
                              process.Close();
                        }
                        // Освобождаем
                        process = null;
                        startinfo = null;
                  }
                  Console.ReadLine();
            }
      }
```



При старте программы 109.exe запускается программа rar.exe и сообщает через консоль, что создаётся архив letter.rar и в него добавляется файл copycat1.cs. Операция выполнена успешно, о чём свидетельствуют OK Done.

Создание командного файла из C#-программы.

Создание текстового файла (bat или cmd)

Для того чтобы создать командный файл необходимо создать поток для записи в файл.

```
StreamWriter s = new StreamWriter("1.cmd", false);
```

где “1.cmd” – это имя создаваемого файла;

false - определяет, требуется ли добавить в файл данные.Если файл существует и значение этого параметра равно false, файл перезаписывается; если не существует, создается новый файл.

После создания потока необходимо записать в файл соответствующие команды. Запись в файл производится командами класса System.Console, такими, как Write() или WriteLine(). Например, создать директорий с именем “1” можно по команде операционной системы md 1. Перед командой надо указать имя созданного потока:

```
s.WriteLine("md 1");
```

Вывод на экран созданного файла:

выведем на экран содержимое нашего командного файла. Для этого создаем поток для чтения из файла и построчно выводим на экран его содержимое.

```
StreamReader r = new StreamReader("1.cmd");
while (!r.EndOfStream)
{
    string str = r.ReadLine();
    Console.WriteLine(str);
}
r.Close();
```

Здесь EndOfStream свойство класса StreamReader, определяющее, находится ли позиция текущего потока в конце потока.

Пример.

В папке C:\Моно\222> размещаем файл 110.cs, в котором предусмотрено создание командного файла операционной системы, содержащего всего одну команду: “md 1”, а затем – чтение этого файла с выводом его содержимого на экран.

```
//Файл 110.cs:
using System;
using System.IO;

public class Show
{

        public static void Main()
        {
          StreamWriter s = new StreamWriter("1.cmd", false);
          s.WriteLine("md 1");
          s.Close();
```

```
            StreamReader r = new StreamReader("1.cmd");
            while (!r.EndOfStream)
            {
                string str = r.ReadLine();
                Console.WriteLine(str);
            }
            r.Close();
    Console.Read();
        }
    }
```

После компиляции получаем файл 110.exe, при исполнении которого на экран выводится созданная команда:

В папке с программой 110.cs появился файл 1.cmd:

Заключение (выводы).

Таким образом, в результате анализа системы команд ОС Windows 2000/XP, прикладных программ Windows и возможностей алгоритмического языка C# установлено, что

1) система команд ОС может быть существенно расширена за счёт использования в качестве внешних команд скриптов (командных файлов) и внешних для ОС программных средств, управляющих внешними программами, такими, как архиваторы, антивирусные средства, программные комплексы для обслуживания алгоритмических языков (например, ассемблера), и др.
2) необходимые для комплексирования программных средств операции, такие, как перехват результатов работы программ, выделение из перехваченных данных сведений, необходимых для дальнейшего использования, соединение данных, полученных из разных источников и передача их программе для дальнейшей обработки, могут быть выполнены, в основном, средствами ОС и дополнительными командными файлами.
3) Языки высокого уровня, такие, как C#, имеют всё необходимое и допускают разработку программ, содержащих низкоуровневые операции, необходимые для комплексирования программных средств.

В ходе анализа возможность комплексирования программных средств подтверждена совершенствованием тестирующей программы; построением структуры каталогов при выявлении длинных имён файлов; принципиальным решением проблемы коррекции неправильных каталогов, содержащих длинные имена.

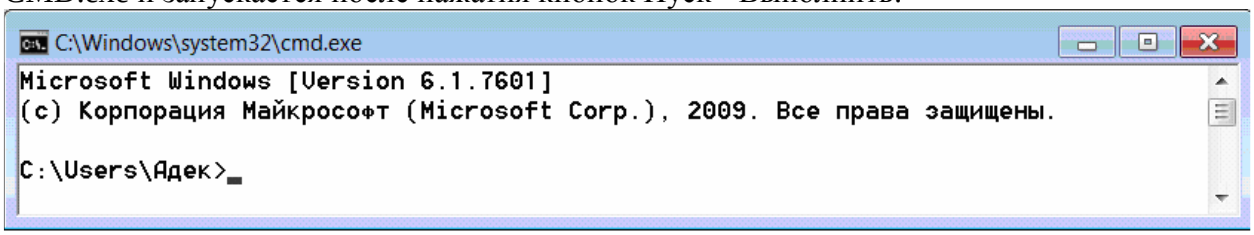
Комплексирование программных средств – это стиль программирования, при котором

программист расширяет возможности алгоритмического языка и значительно сокращает сроки создания программ.

Алгоритмические языки. Практикум по основам программирования на C#.

Командный процессор cmd и его дополнение компилятором csc.

В Windows используется несколько разных командных процессоров. Основной имеет имя CMD.exe и запускается после нажатия кнопок Пуск - Выполнить:



По команде help он распечатывает свою систему команд - систему команд операционной системы Windows. Команда csc в ней отсутствует.

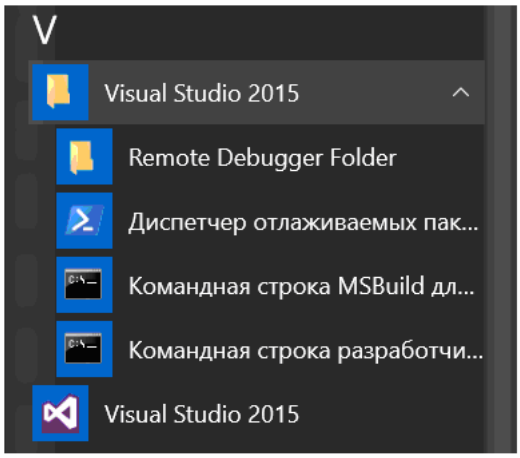
Visual Studio имеет несколько компиляторов, запускаемых разными процессорами. Кроме компилятора, запускаемого из основного меню пакета, C#-программы можно обрабатывать с помощью внешнего компилятора этого пакета. Он строится на основе командного процессора cmd.exe, дополнительно обработанного специальным bat-файлом, находящимсяпо адресу:

C:\Program Files (x86)\Microsoft Visual Studio 4.0\Common7\Tools\VsDevCmd.bat.

Чтобы его найти, надо зайти на диск C:, пройти через все папки по адресам:

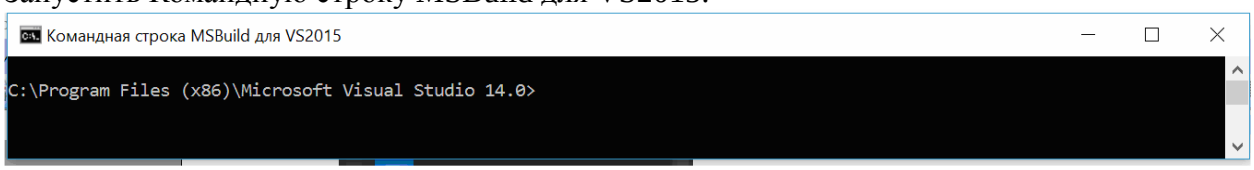
C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\Tools\ и запустить файл VsDevCmd.bat.

Затем нажать Пуск –> Все приложения, найти Visual Studio 2015:

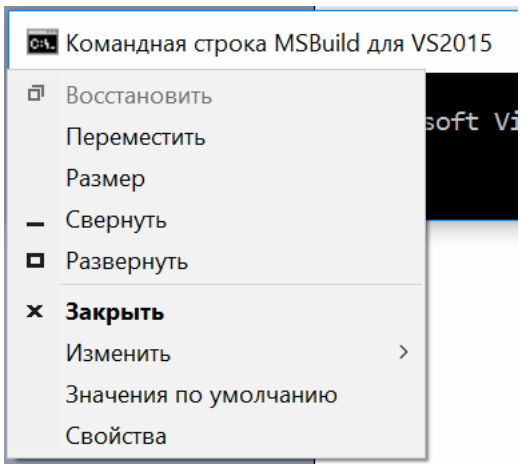


- искомая программа

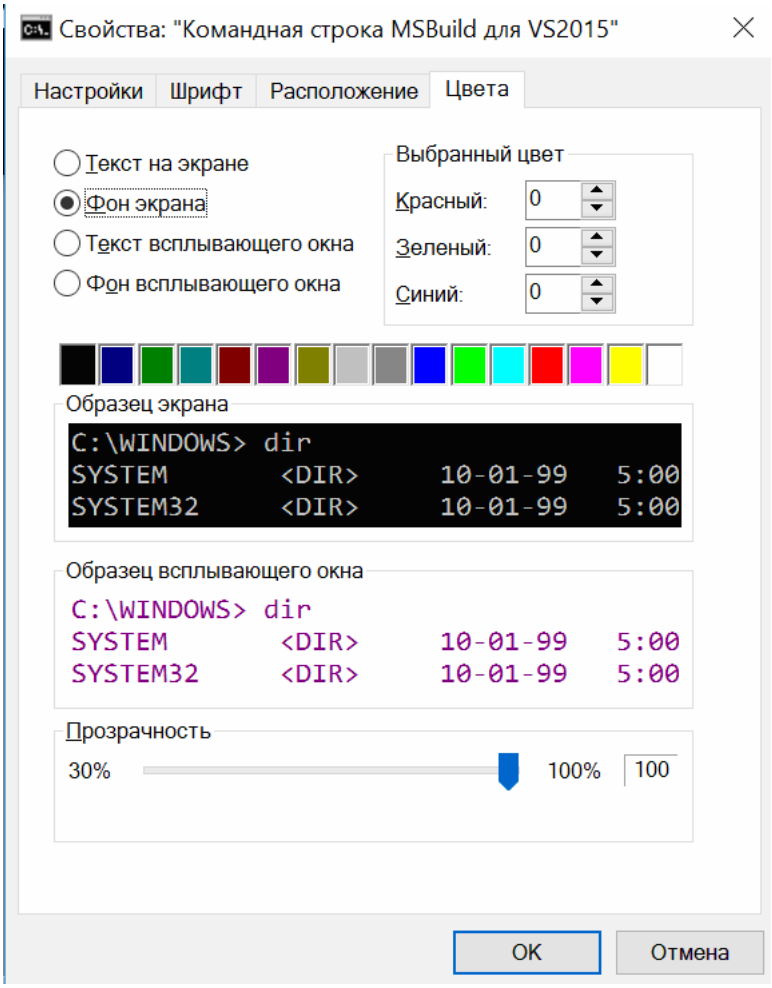
Запустить Командную строку MSBuild для VS2015:



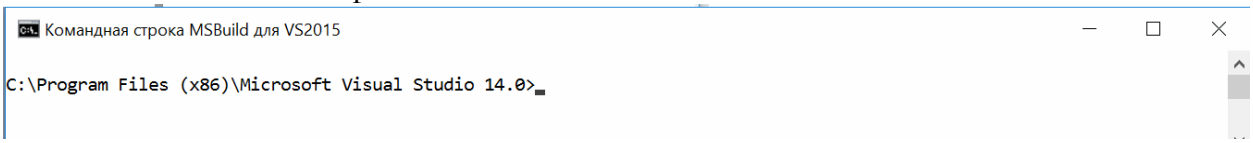
Нажать на иконку в верхнем левом углу окна командного процессора:



Выбрать Свойства:



Изменить цвета текста и фона:



Просмотреть состав команд получившегося командного процессора на 64-битном компьютере, можно по команде help:

C:\Program Files (x86)\Microsoft Visual Studio 14.0>help

Система команд cmd после обработки bat-файлом.

Состав команд:

ASSOC Вывод либо изменение сопоставлений по расширениям имен файлов.

ATTRIB Отображение и изменение атрибутов файлов.
BREAK Включение и выключение режима обработки комбинации клавиш CTRL+C.
BCDEDIT Задает свойства в базе данных загрузки для управления начальной загрузкой.
CACLS Отображение и редактирование списков управления доступом (ACL) к файлам.
CALL Вызов одного пакетного файла из другого.
CD Вывод имени либо смена текущей папки.
CHCP Вывод либо установка активной кодовой страницы.
CHDIR Вывод имени либо смена текущей папки.
CHKDSK Проверка диска и вывод статистики.
CHKNTFS Отображение или изменение выполнения проверки диска во время загрузки.
CLS Очистка экрана.
CMD Запуск еще одного интерпретатора командных строк Windows.
COLOR Установка цветов переднего плана и фона, используемых по умолчанию.
COMP Сравнение содержимого двух файлов или двух наборов файлов.
COMPACT Отображение и изменение сжатия файлов в разделах NTFS.
CONVERT Преобразование дисковых томов FAT в NTFS. Нельзя выполнить преобразование текущего активного диска.
COPY Копирование одного или нескольких файлов в другое место.
DATE Вывод либо установка текущей даты.
DEL Удаление одного или нескольких файлов.
DIR Вывод списка файлов и подпапок из указанной папки.
DISKCOMP Сравнение содержимого двух гибких дисков.
DISKCOPY Копирование содержимого одного гибкого диска на другой.
DISKPART Отображение и настройка свойств раздела диска.
DOSKEY Редактирование и повторный вызов командных строк; создание макросов.
DRIVERQUERY Отображение текущего состояния и свойств драйвера устройства.
ECHO Вывод сообщений и переключение режима отображения команд на экране.
ENDLOCAL Конец локальных изменений среды для пакетного файла.
ERASE Удаление одного или нескольких файлов.
EXIT Завершение работы программы CMD.EXE (интерпретатора командных строк).
FC Сравнение двух файлов или двух наборов файлов и вывод различий между ними.
FIND Поиск текстовой строки в одном или нескольких файлах.
FINDSTR Поиск строк в файлах.
FOR Запуск указанной команды для каждого из файлов в наборе.
FORMAT Форматирование диска для работы с Windows.
FSUTIL Отображение и настройка свойств файловой системы.
FTYPE Вывод либо изменение типов файлов, используемых при сопоставлении по расширениям имен файлов.
GOTO Передача управления в отмеченную строку пакетного файла.
GPRESULT Отображение информации о групповой политике для компьютера или пользователя.
GRAFTABL Позволяет Windows отображать расширенный набор символов в графическом режиме.
HELP Выводит справочную информацию о командах Windows.
ICACLS Отображение, изменение, архивация или восстановление

списков ACL для файлов и каталогов.
IF Оператор условного выполнения команд в пакетном файле.
LABEL Создание, изменение и удаление меток тома для дисков.
MD Создание папки.
MKDIR Создание папки.
MKLINK Создание символических и жестких ссылок
MODE Конфигурирование системных устройств.
MORE Последовательный вывод данных по частям размером в один экран.
MOVE Перемещение одного или нескольких файлов из одной папки в другую.
OPENFILES Отображение файлов, открытых на общей папке удаленным пользователем.
PATH Отображает или устанавливает путь поиска исполняемых файлов.
PAUSE Приостанавливает выполнение пакетного файла и выводит сообщение.
POPD Восстанавливает предыдущее значение активной папки, сохраненное с помощью команды PUSHD.
PRINT Выводит на печать содержимое текстового файла.
PROMPT Изменяет приглашение в командной строке Windows.
PUSHD Сохраняет значение активной папки и переходит к другой папке.
RD Удаляет папку.
RECOVER Восстанавливает данные, которые можно прочитать, с плохого или поврежденного диска.
REM Помещает комментарии в пакетные файлы и файл CONFIG.SYS.
REN Переименовывает файлы или папки.
RENAME Переименовывает файлы или папки.
REPLACE Замещает файлы.
RMDIR Удаление папки.
ROBOCOPY Улучшенное средство копирования файлов и деревьев каталогов
SET Показывает, устанавливает и удаляет переменные среды Windows.
SETLOCAL Начинает локализацию изменений среды в пакетном файле.
SC Отображает и настраивает службы (фоновые процессы).
SCHTASKS Выполняет команды и запускает программы по расписанию.
SHIFT Изменение положения (сдвиг) подставляемых параметров для пакетного файла.
SHUTDOWN Локальное или удаленное выключение компьютера.
SORT Сортировка ввода.
START Выполнение программы или команды в отдельном окне.
SUBST Назначение заданному пути имени диска.
SYSTEMINFO Вывод сведений о системе и конфигурации компьютера.
TASKLIST Отображение всех выполняемых задач, включая службы.
TASKKILL Прекращение или остановка процесса или приложения.
TIME Вывод и установка системного времени.
TITLE Назначение заголовка окна для текущего сеанса интерпретатора командных строк CMD.EXE.
TREE Графическое отображение структуры каталогов диска или папки.

TYPE Вывод на экран содержимого текстовых файлов.
VER Вывод сведений о версии Windows.
VERIFY Установка режима проверки правильности записи файлов на диск.
VOL Вывод метки и серийного номера тома для диска.
XCOPY Копирование файлов и деревьев каталогов.
WMIC Вывод сведений WMI в интерактивной среде.

Для получения сведений об определенной команде наберите HELP <имя команды> - дополнительные сведения о программах будут выведены в описании программ из данного списка.

Параметры компилятора Visual C#.

Однако команда csc в данном списке отсутствует, так как в нём перечислены только основные команды Windows. При обработке процессора cmd bat-файлом, в процессор были добавлены дополнительные команды. Чтобы убедиться в этом, надо запросить:

csc /?

Ответ получим такой: Компилятор Microsoft (R) Visual C# версии 1.3.1.60616c Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Параметры компилятора Visual C#

- ВЫХОДНЫЕ ФАЙЛЫ -

Параметры компилятора	Пояснение
/out:<файл>	Указать имя выходного файла (по умолчанию базовое имя файла с классом main или имя первого файла)
/target:exe	Выполнить сборку консольного исполняемого файла - по умолчанию (краткая форма: /t:exe)
/target:winexe	Выполнить сборку исполняемого файла Windows (крат кая форма: /t:winexe)
/target:library	Выполнить сборку библиотеки (краткая форма: /t:li brary) /target:module Выполнить сборку модуля, который может быть добав лен в другую сборку (краткая форма: /t:module)
/target:appcontainerexe	Выполнить сборку исполняемого файла Appcontainer (краткая форма: /t:appcontainerexe)
/target:winmdobj	Выполнить сборку промежуточного файла среды выполнения Windows, используемого в WinMDExp (краткая форма: /t:winmd obj)
/doc:<файл>	Создаваемый файл XML-документации
/platform:<строка>	Ограничить платформы, на которых может выполняться этот код: x86, Itanium, x64, arm, anycpu32bitpreferred или anycpu. Значение по умолчанию - anycpu.

- ВХОДНЫЕ ФАЙЛЫ -

Параметры компилятора	Пояснение
/recurse:<подстановка>	Включить все файлы в текущем каталоге и подкаталогах в соответствии с заданным подстановочным знаком
/reference:<псевдоним>=<файл>	Сослаться на метаданные из заданного файла сборок с помощью определенного псевдонима (краткая форма: /r)
/reference:<файлы>	Сослаться на метаданные из заданных файлов сборок (краткая форма: /r)
/addmodule:<файлы>	Компоновать указанные модули со сборкой
/link:<файлы>	Внедрять метаданные из указанных файлов сборок взаимодействия (краткая форма: /l)
/analyzer:<файлы>	Запускать анализаторы из этой сборки (краткая форма: /a)
/additionalfile:<файлы>	Дополнительные файлы, которые не оказывают прямог о влияния на создание кода, однако могут использоваться анализаторами для вывода ошибок или предупреждений

- РЕСУРСЫ -

Параметры компилятора	Пояснение
/win32res:<файл>	Задать файл ресурсов Win32 (RES-файл)
/win32icon:<файл>	Использовать этот значок для вывода
/win32manifest:<файл>	Задать файл манифеста Win32 (XML-файл)

/nowin32manifest	Не включать манифест Win32 по умолчанию
/resource:<сведения о ресурсе>	Внедрить указанный ресурс (краткая форма: /res)
/linkresource:<сведения о ресурсе>	Скомпоновать указанный ресурс вместе с этой сборкой (краткая форма: /linkres), где формат сведений о ресурсах - <файл>[,<имя строки>[,public|private]]

- НАПИСАНИЕ КОДА -

Параметры компилятора	Пояснение
/debug[+|-]	Выдать отладочную информацию
/debug:{full|pdbonly|portable}	Задать тип отладки (по умолчанию - full) и позволить подключить отладчик к выполняющимся программам), portable представляет собой кроссплатформенный формат
/optimize[+|-]	Включить оптимизацию (краткая форма: /o)
/deterministic	Создать детерминированную сборку (включая метку времени и GUID версии модуля)

- ОШИБКИ И ПРЕДУПРЕЖДЕНИЯ -

Параметры компилятора	Пояснение
/warnaserror[+|-]	Записывать все предупреждения как ошибки
/warnaserror[+|-]:<предупр.>	Записывать отдельные предупреждения как ошибки
/warn:<n>	Задать уровень предупреждений (0-4) (краткая форм а: /w)
/nowarn:<предупреждения>	Отключить указанные предупреждения
/ruleset:<файл>	Указать файл набора правил, отключающий определенные диагностические операции
/errorlog:<файл>	Указать файл для ведения журнала всех диагностических данных компилятора и анализатора
/reportanalyzer	Сообщить дополнительные сведения об анализаторе, Например время выполнения

- ЯЗЫК -

Параметры компилятора	Пояснение
/checked[+|-]	Создать проверки переполнений
/unsafe[+|-]	Допускать "небезопасный" код
/define:<список символов>	Определить символы условной компиляции (краткая форма: /d)
/langversion:<строка>	Указать режим языковой версии: ISO-1, ISO-2, 3, 4, 5, 6 или по умолчанию

- БЕЗОПАСНОСТЬ -

Параметры компилятора	Пояснение
/delaysign[+|-]	Использовать отложенную подпись для сборки, используя только открытую часть ключа строгого имени
/keyfile:<файл>	Указать файл ключей строгого имени
/keycontainer:<строка>	Указать контейнер ключей строгого имени
/highentropyva[+|-]	Включить ASLR с высокой энтропией

- ПРОЧЕЕ -

Параметры компилятора	Пояснение
@<file>	Считать файл ответов с дополнительными параметрами
/help	Отображать это сообщение об использовании (краткая форма: /?)
/nologo	Запрещать отображение сообщения компилятора об ав торских правах
/noconfig	Не включать файл CSC.RSP автоматически
/parallel[+|-]	Параллельная сборка

- ДОПОЛНИТЕЛЬНО -

Параметры компилятора	Пояснение
/baseaddress:<адрес>	Базовый адрес библиотеки, для которой выполняется сборка
/bugreport:<файл>	Создать файл "Отчет об ошибке"

/checksumalgorithm:<алгоритм>	Задать алгоритм расчета контрольной суммы исходного файла, хранимой в PDB-файле. Поддерживаемые значения: SHA1 (по умолчанию) или SHA256
/codepage:<n>	Задать кодовую страницу для использования при открытии исходных файлов
/utf8output	Выводить сообщения компилятора в кодировке UTF-8
/main:<тип>	Указать тип, содержащий точку входа (все другие возможные точки входа пропускаются) (краткая форма: /m)
/fullpaths	Компилятор создает полные пути
/filealign:<n>	Задать выравнивание для секций выходных файлов
/pathmap:<K1>=<V1>,<K2>=<V2>,...	Задать сопоставление для выходных данных имен исходного пути по компилятору
/pdb:<файл>	Задать имя файла для отладочной информации (по умолчанию используется имя выходного файла с расширением PD B)
/errorendlocation	Выходные строка и столбец конечного расположения каждой ошибки
/preferreduilang	Указать имя предпочтительного языка вывода
/nostdlib[+|-]	Не обращаться к стандартной библиотеке (mscorlib. dll)
/subsystemversion:<строка>	Задать версию подсистемы этой сборки
/lib:<список файлов>	Задать дополнительные каталоги для поиска ссылок
/errorreport:<строка>	Указать способ обработки внутренних ошибок компилятора: prompt, send, queue или none. По умолчанию используется queue
/appconfig:<файл>	Указать файл конфигурации приложения, содержащий параметры привязки сборки
/moduleassemblyname:<строка>	Имя сборки, частью которой будет являться данный модуль
/modulename:<строка>	Задать имя исходного модуля

Примеры команд командной строки для компилятора C#

- Компиляция файла File.cs в файл **File.exe**:

csc File.cs

- Компиляция файла File.cs в файл **File.dll**:

csc /target:library File.cs

- Компиляция файла File.cs и создание файла **My.exe**:

csc /out:My.exe File.cs

- Компиляция **всех** файлов C# в текущем каталоге с оптимизацией и определением символа DEBUG. Результат File2.exe:

csc /define:DEBUG /optimize /out:File2.exe *.cs

- Компиляция всех файлов C# в текущем каталоге с созданием версии отладки файла File2.dll. Отключение отображения логотипа и предупреждений:

csc /target:library /out:File2.dll /warn:0 /nologo /debug *.cs

- Компиляция всех файлов C# в текущем каталоге в файл Something.xyz (**библиотека DLL**):

csc /target:library /out:Something.xyz *.cs

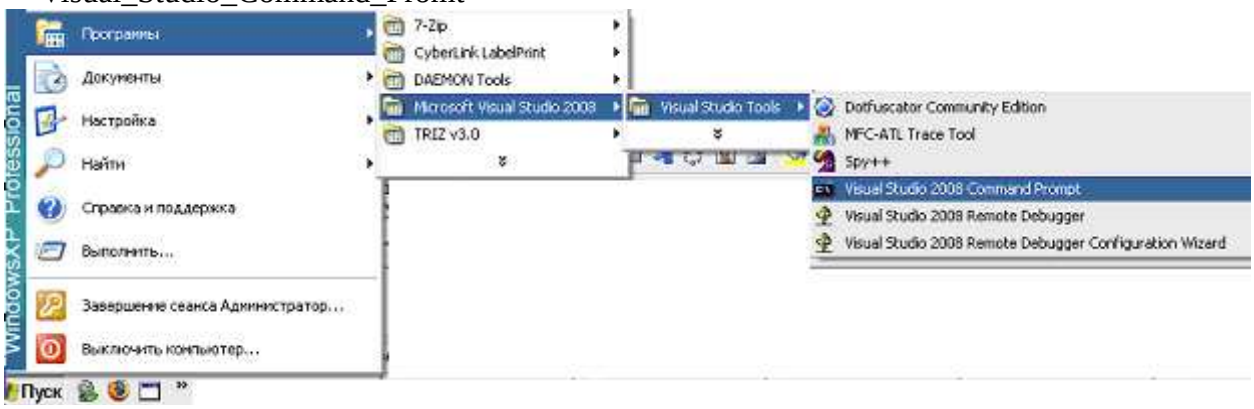
Запуск и выполнение C#-программ.

Первый способ выполнения cs-программы.

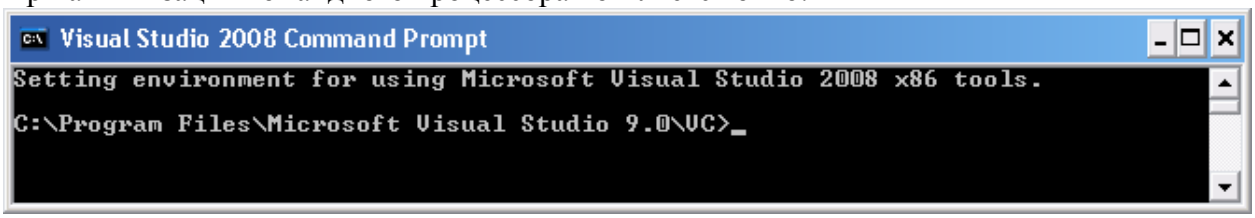
Подготовка к работе командного процессора.

Командный процессор, подготовленный для работы с Visual Studio находится по следующему пути:

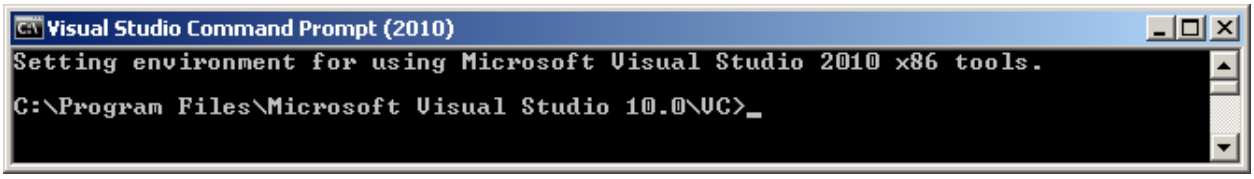
Пуск –> Программы –> Microsoft_Visual_Studio_–> Visual_Studio_Tools –>
-> Visual_Studio_Command_Promt



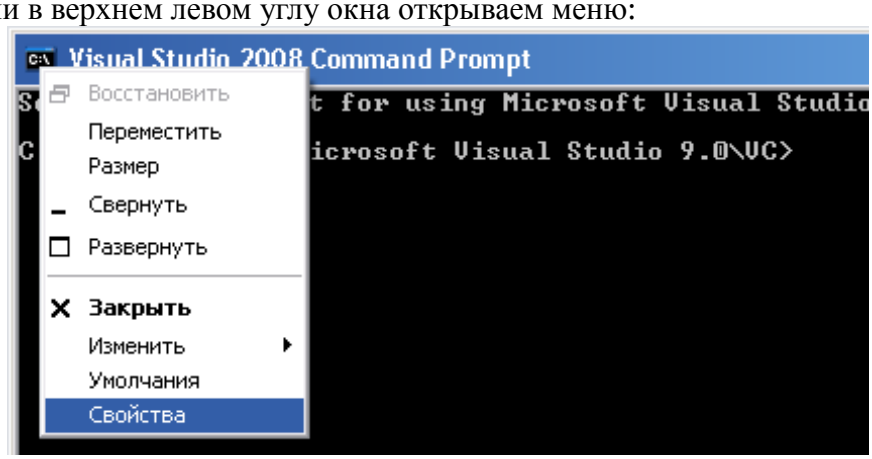
При активизации командного процессора появляется окно:



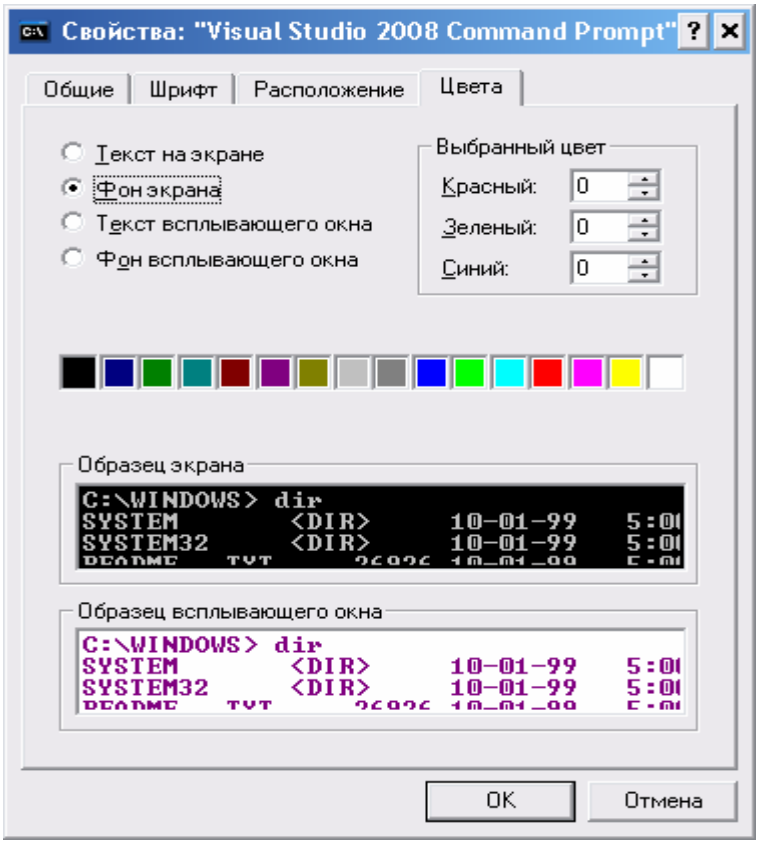
или соответственно:



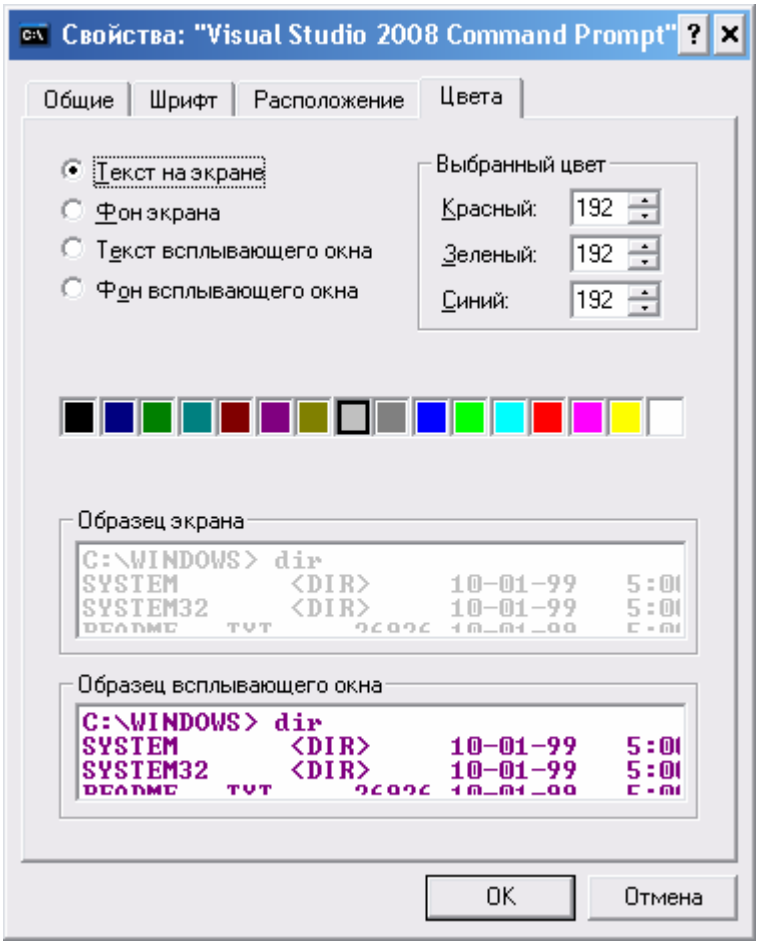
Желательно это окно преобразовать, изменив цвета фона и символов. Для этого правой кнопкой мыши в верхнем левом углу окна открываем меню:



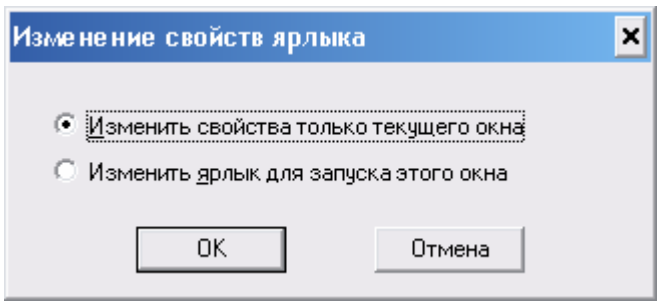
И входим в пункт Свойства:
Изменяем цвета фона и символов: фон экрана:



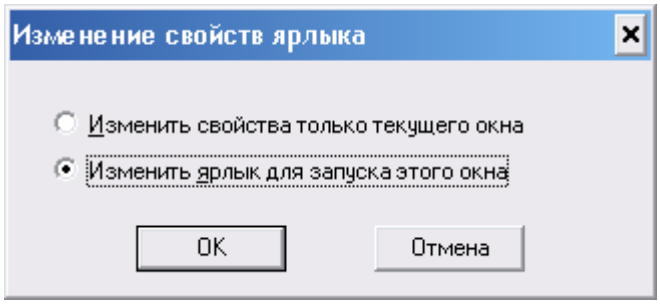
Текст на экране:



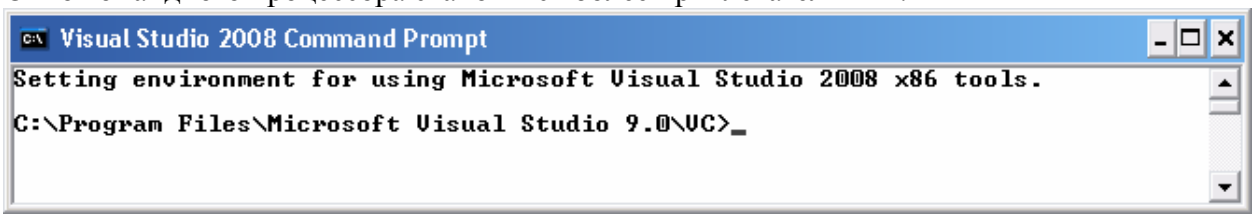
После установки цвета появляется окно:



Выбираем:



Окно командного процессора становится более привлекательным:

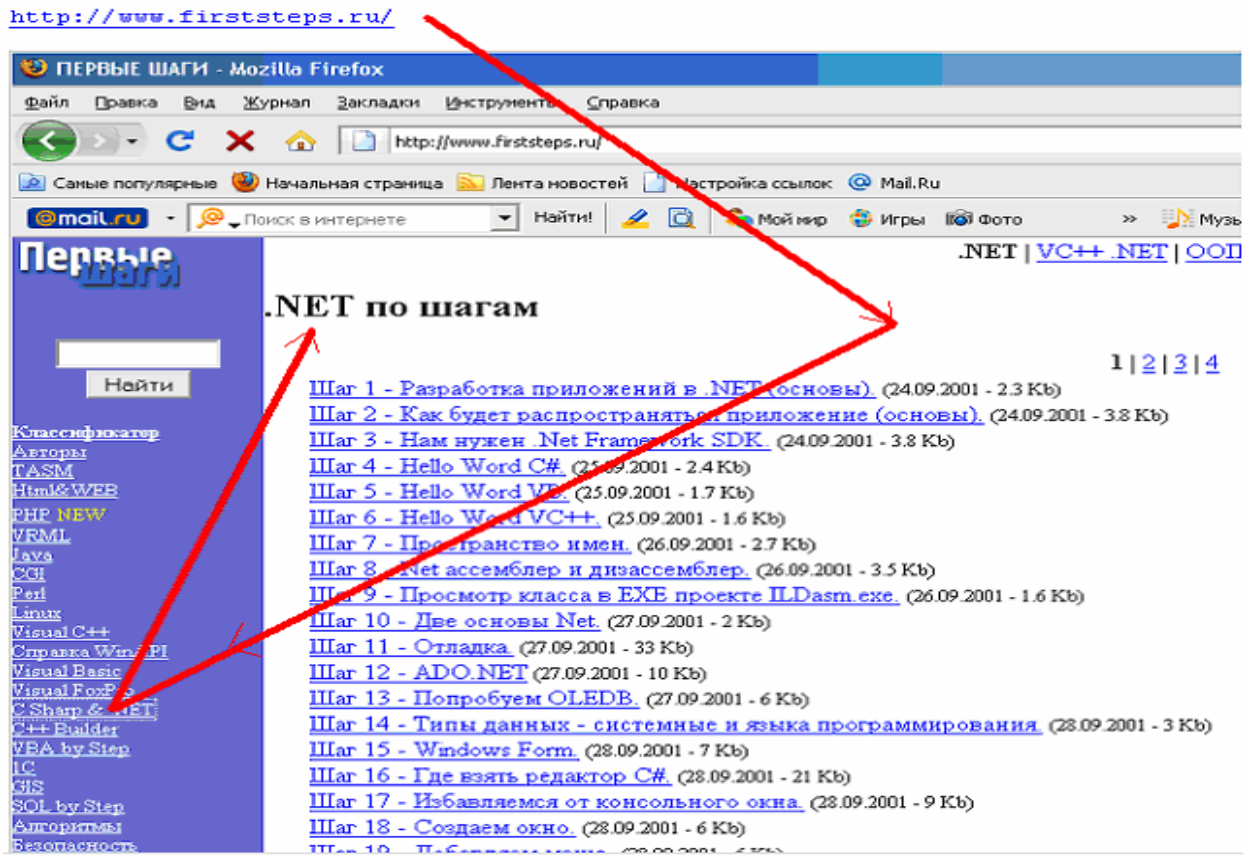


Перейти в корень диска можно по команде cd \.
Для смены диска надо указать его имя: Например d: <ET>.

Примеры C#-программ на сайтах Интернет.

1.

http://www.firststeps.ru/



2. http://subscribe.ru/catalog/comp.soft.prog.csharplessons?pos=2
Это сайт «Уроки по C. Рассылки сайта **progs.biz** Выпуск No 103»
Начало > C# > Windows > Урок NN

Структура C# - программы.

1. Первая программа.

```
using System:
namespace ConsoleApplicationl
{
  class Classl
  {
   static void Main()
   {
    Console.WriteLine( " Да здравствует Кот Матроскин" );
    Console.read();
   }
  }
}
```

Для исполнения этой программы в корне доступного диска создадим папку с очень простым названием, например, на диске d: создадим папку 1.В папке 1 создадим текстовый файл 1.txt, сохраним в нём первую программу и переименуем его в файл 1.cs. После переименования должна измениться иконка файла.
Затем настраиваем командный процессор на работу с папкой 1 диска d:.
Вызываем компилятор csc и передаём ему файл 1.cs. Если компиляция пройдёт успешно в папке 1 появится новый файл 1.exe.

2. Что должны содержать C# - программы.

Текст программы.

```
public static void Main(string[] args) {
    Console.Write("Hola ");
    Console.WriteLine("Mundo!");
    Console.WriteLine("What is your name: ");
    String name = Console.ReadLine();
    Console.Write("Buenos Dias, ");
    Console.Write(name);
    Console.WriteLine("!");
}
```

Компиляция этого текста:

```
C:\12>csc 1.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.1433
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

1.cs(1,15): error CS1518: Expected class, delegate, enum, interface, or
struct
1.cs(1,32): error CS1001: Identifier expected
1.cs(1,34): error CS1518: Expected class, delegate, enum, interface, or
struct

C:\12>
```

Компилятор фиксирует наличие ошибок.

Добавляем к программе оператор class:

```
class A
{
```

```
public static void Main(string[] args) {
    Console.Write("Hola ");
    Console.WriteLine("Mundo!");
    Console.WriteLine("What is your name: ");
    String name = Console.ReadLine();
    Console.Write("Buenos Dias, ");
    Console.Write(name);
    Console.WriteLine("!");
}
}
```

В результате компиляции получим:

```
C:\12>csc 1.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.1433
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

1.cs(4,5): error CS0103: The name 'Console' does not exist in the current
        context
1.cs(5,5): error CS0103: The name 'Console' does not exist in the current
        context
1.cs(6,5): error CS0103: The name 'Console' does not exist in the current
        context
1.cs(7,5): error CS0246: The type or namespace name 'String' could not be
found
        (are you missing a using directive or an assembly reference?)
1.cs(7,19): error CS0103: The name 'Console' does not exist in the current
        context
1.cs(8,5): error CS0103: The name 'Console' does not exist in the current
        context
1.cs(9,5): error CS0103: The name 'Console' does not exist in the current
        context
1.cs(10,5): error CS0103: The name 'Console' does not exist in the current
        context
```

опять ошибки.

Добавляем ещё один оператор – using System;

```
using System;
class A
{
public static void Main(string[] args) {
    Console.Write("Hola ");
    Console.WriteLine("Mundo!");
    Console.WriteLine("What is your name: ");
    String name = Console.ReadLine();
    Console.Write("Buenos Dias, ");
    Console.Write(name);
    Console.WriteLine("!");
}
}
```

Компиляция проходит успешно:

```
C:\12>csc 1.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.1433
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.
```

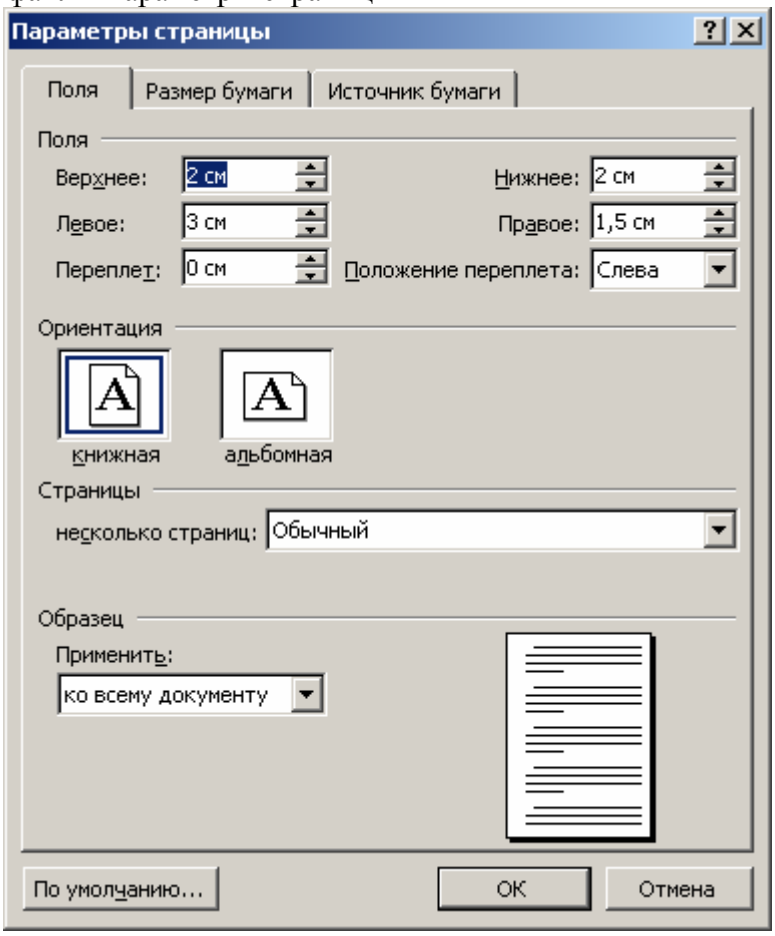
Программу можно выполнить:

```
C:\12>1
Hola Mundo!
What is your name:
Kola
Buenos Dias, Kola!
C:\12>
```

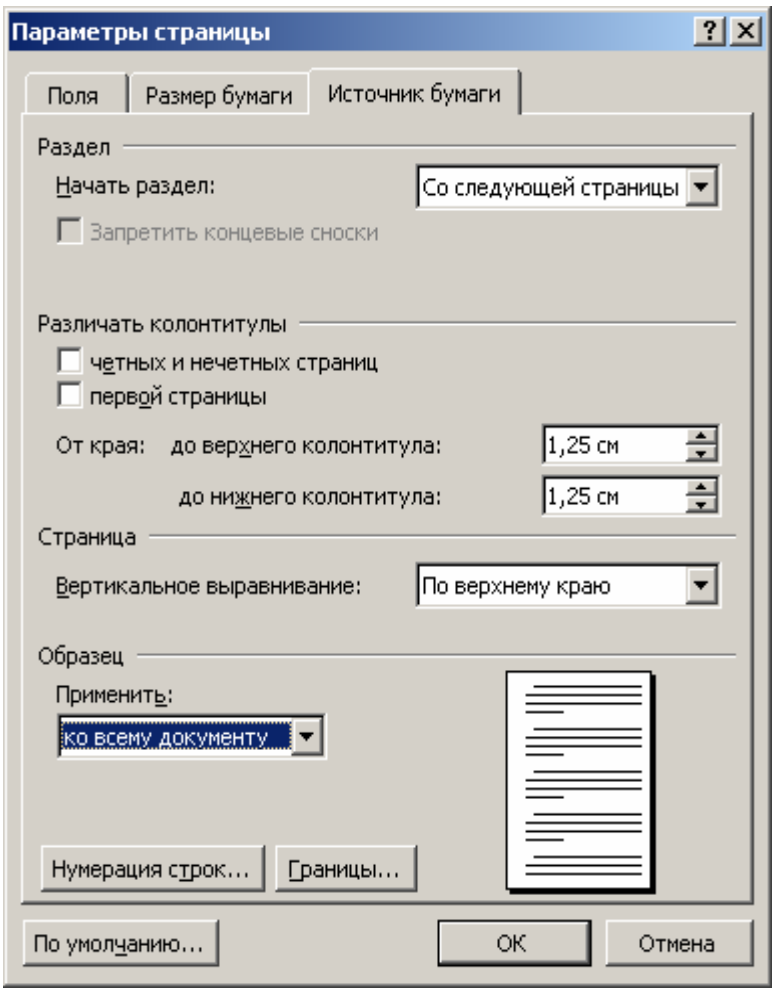
Нумерация строк программы в Word.

При составлении отчёта удобно иметь программу с перенумерованными строками.
Для нумерации строк надо:

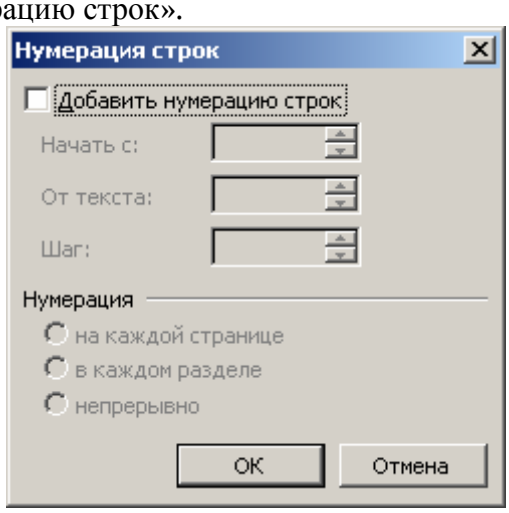
1. Выделить необходимые строки
2. Перейти в файл – параметры страницы



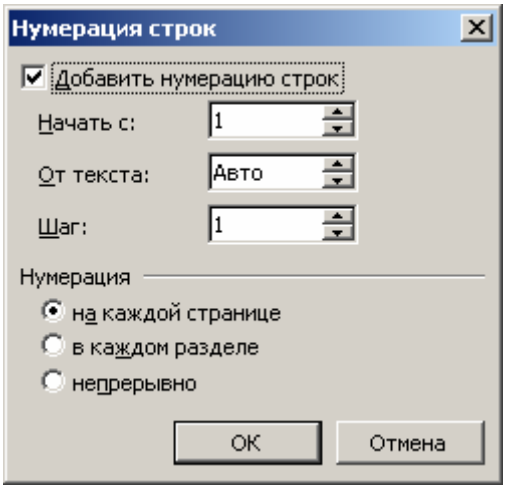
В нижней части приведен Образец – «ко всему документу», или «до конца документа», а при наличии выделенного текста – и «к выделенному тексту».
Вкладка «Источник бумаги» содержит кнопку «Нумерация строк»:



При нажатии на эту кнопку появляется окно, в котором есть возможность включить функцию «Добавить нумерацию строк».



После включения этой функции изменяется внешний вид этого окна:



Появляется возможность установить, с какой цифры начать нумерацию, с каким шагом, на каждой странице, в каждом разделе или непрерывно по всему выделенному тексту.

Плохо только, что эта нумерация имеет временный характер – нельзя её оставить и продолжать работу над текстом со ссылками на номера строк. Кроме того, при выделении фрагмента меняется структура текста – в него автоматически включается перенос выделенного фрагмента на новую страницу.

Единственный способ – через буфер обмена вставить часть текста в Paint, и затем скопировать его в графическом виде для вставки в текст:

```
1   using System;
2   using System.Drawing;
3   using System.Collections;
4   using System.ComponentModel;
5   using System.Windows.Forms;
6   using System.Data;
7
8   namespace FirstProgram_
9   {
10          /// <summary>
11          /// Summary description for Form1.
12          /// </summary>
13          public class Form1 : System.Windows.Forms.Form
14          {
15                  private System.Windows.Forms.Label label1;
16                  private System.Windows.Forms.Button button1;
17                  /// <summary>
18                  /// Required designer variable.
19                  /// </summary>
20                  private System.ComponentModel.Container components = null;
21
22                  public Form1()
23                  {
24                          //
25                          // Required for Windows Form Designer support
26                          //
27                          InitializeComponent();
28
29                          //
30                          // TODO: Add any constructor code after InitializeComponent
31  call
32                          //
33                  }
34
35                  /// <summary>
36                  /// Clean up any resources being used.
37                  /// </summary>
38                  protected override void Dispose( bool disposing )
39                  {
```

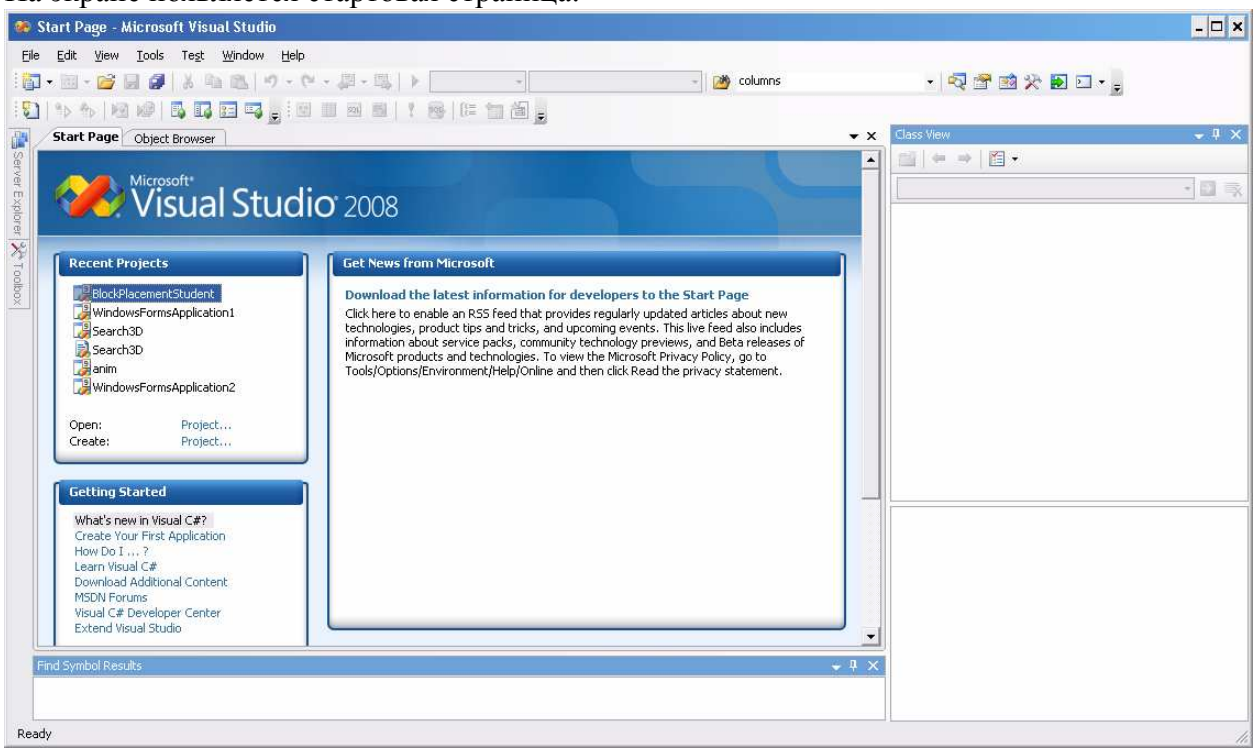
```
40                      if( disposing )
41                      {
42                              if (components != null)
43                              {
44                                      components.Dispose();
45                              }
46                      }
47                      base.Dispose( disposing );
48              }
49
50              #region Windows Form Designer generated code
51              /// <summary>
52              /// Required method for Designer support - do not modify
53              /// the contents of this method with the code editor.
54              /// </summary>
55              private void InitializeComponent()
56              {
57                      this.label1 = new System.Windows.Forms.Label();
58                      this.button1 = new System.Windows.Forms.Button();
59                      this.SuspendLayout();
60                      //
61                      // label1
62                      //
63                      this.label1.Location = new System.Drawing.Point(48, 32);
64                      this.label1.Name = "label1";

65                      this.label1.Size = new System.Drawing.Size(200, 96);
66                      this.label1.TabIndex = 0;
67                      this.label1.Text = "Здравствуйте, меня зовут \"Машина\".";
68                      //
69                      // button1

70                      //
71                      this.button1.Location = new System.Drawing.Point(128, 176);
72                      this.button1.Name = "button1";
73                      this.button1.Size = new System.Drawing.Size(128, 32);
74                      this.button1.TabIndex = 1;
75                      this.button1.Text = "Выход";
76                      this.button1.Click += new
77  System.EventHandler(this.button1_Click);
78                      //
79                      // Form1
80                      //
81                      this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
82                      this.ClientSize = new System.Drawing.Size(292, 273);
83                      this.Controls.Add(this.button1);
84                      this.Controls.Add(this.label1);
85                      this.Name = "Form1";
86                      this.Text = "Оконце";
87                      this.ResumeLayout(false);
88
89              }
90              #endregion
91
92              /// <summary>
93              /// The main entry point for the application.
94              /// </summary>
95              [STAThread]
96              static void Main()
97              {
98                      Application.Run(new Form1());
99              }
100
101             private void button1_Click(object sender, System.EventArgs e)
102             {
103                     Close();
104             }
105     }
106 }
```

Теперь можно давать пояснения к программе со ссылкой на соответствующие команды.

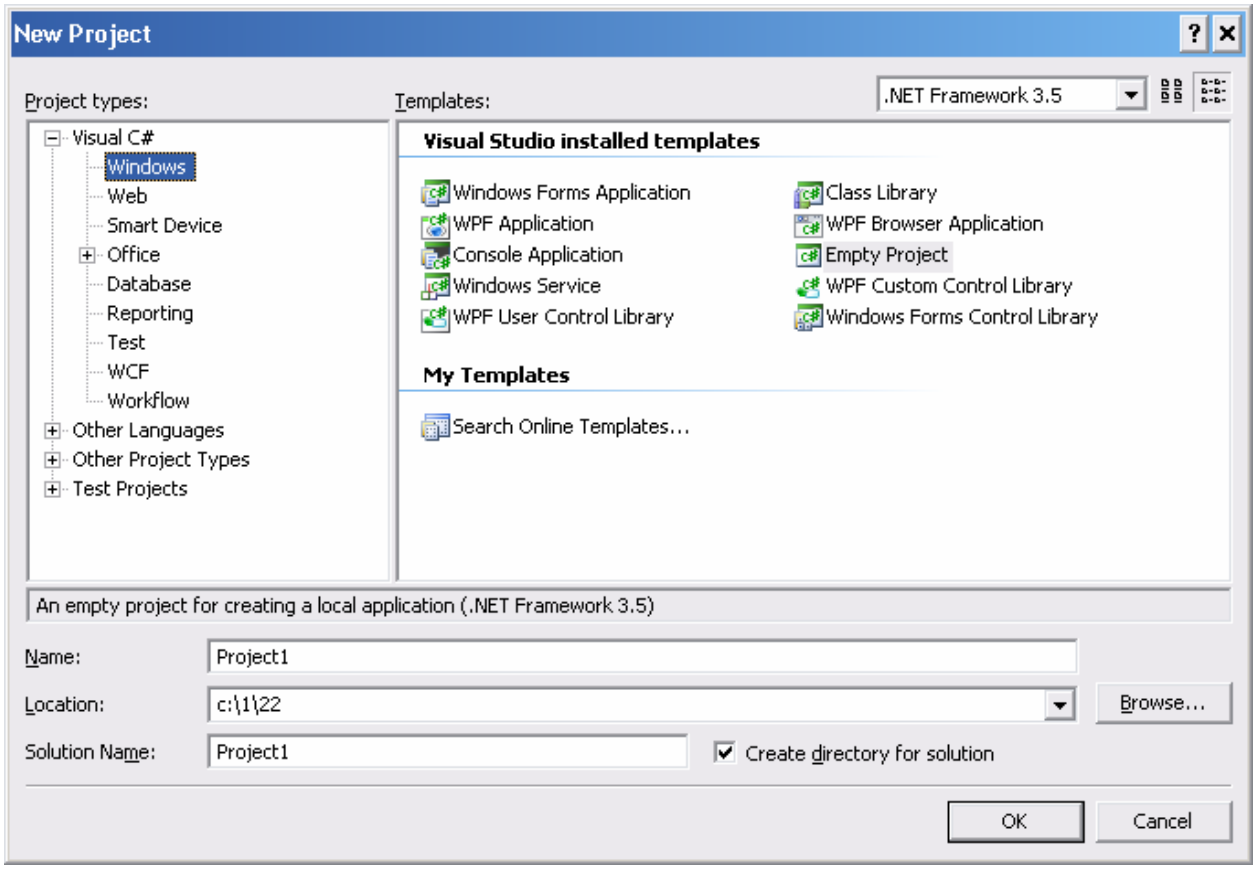
2 способ выполнения cs-программы.

Вход в Microsoft Visual Studio .NET производится через Пуск -> Программы -> Microsoft Visual Studio .NET -> Microsoft Visual Studio .NET, или двойным щелчком по ярлычку:

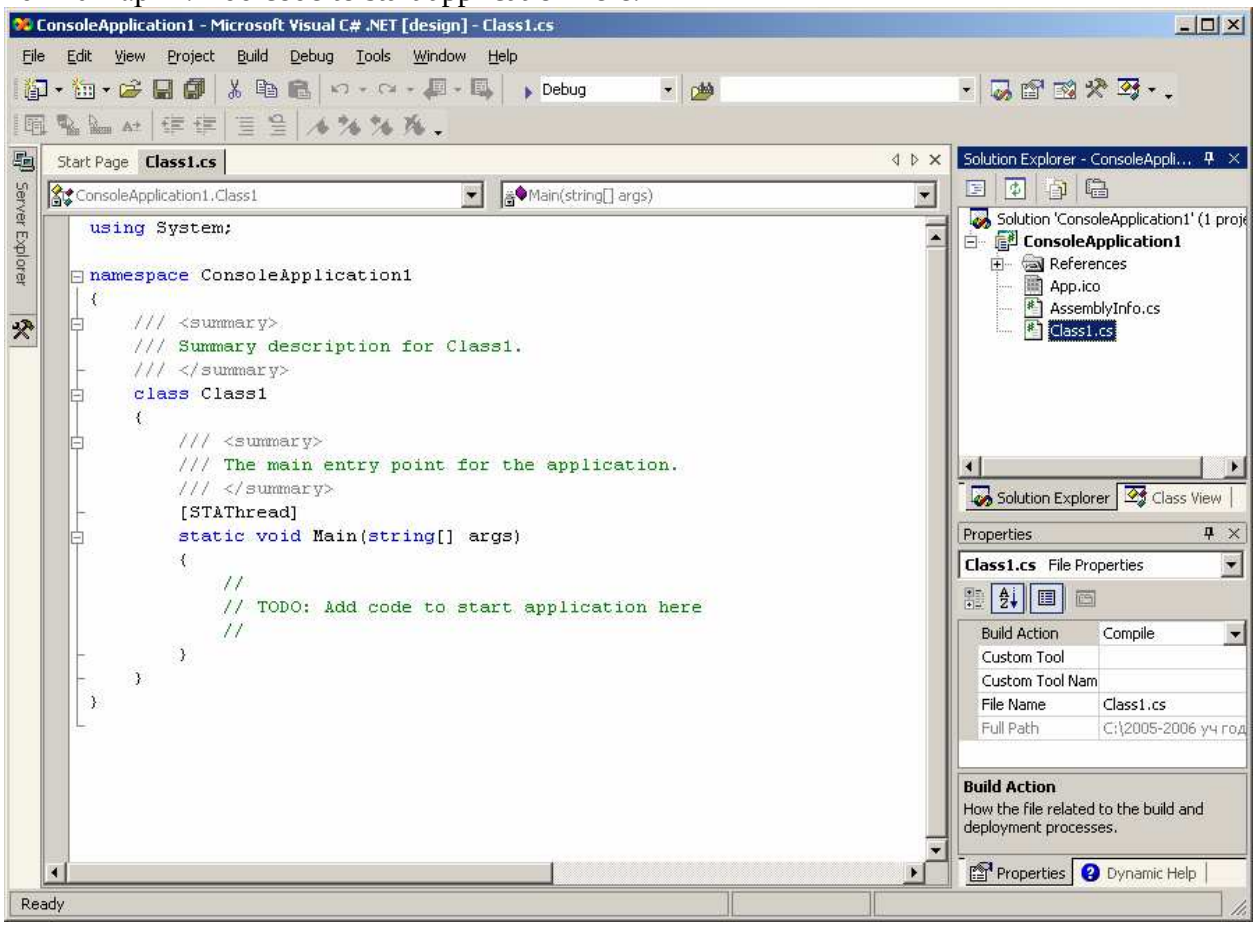
На экране появляется стартовая страница:



После запуска Create Project появится окно, уточняющее тип проекта:



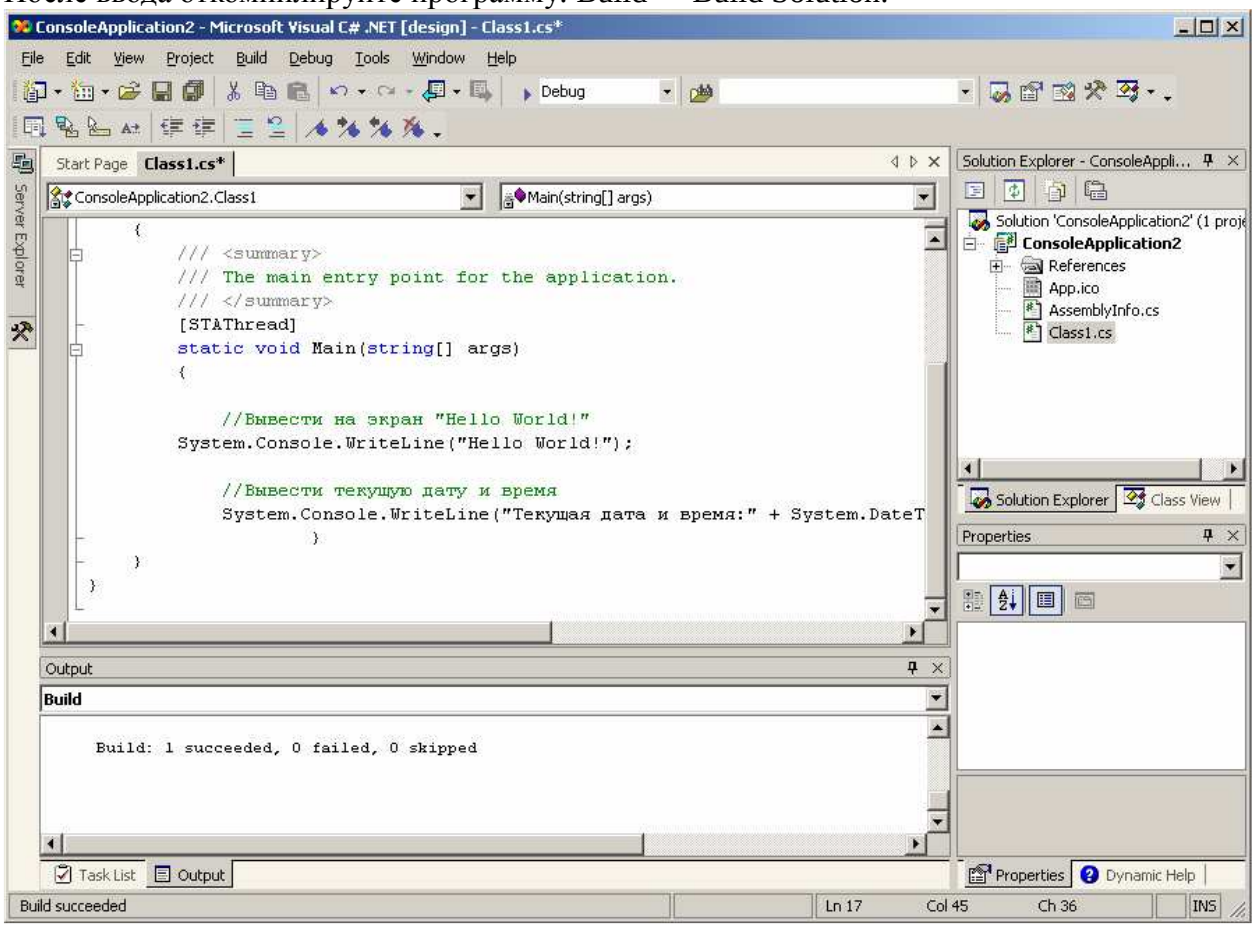
При входе в Microsoft Visual Studio .NET выбрать Console Application.
В окне Microsoft Visual Studio появится заготовка программы, в которой виден комментарий: Add code to start application here.



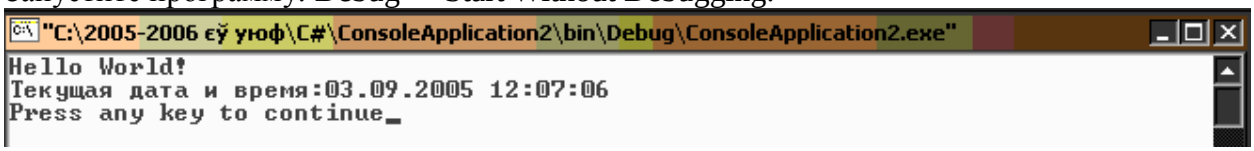
Введите вместо комментария следующие строки:

```
            //Вывести на экран "Hello World!"
        System.Console.WriteLine("Hello World!");

            //Вывести текущую дату и время
            System.Console.WriteLine("Текущая дата и время:" +
System.DateTime.Now);
```

После ввода откомпилируйте программу: Build -> Build Solution:

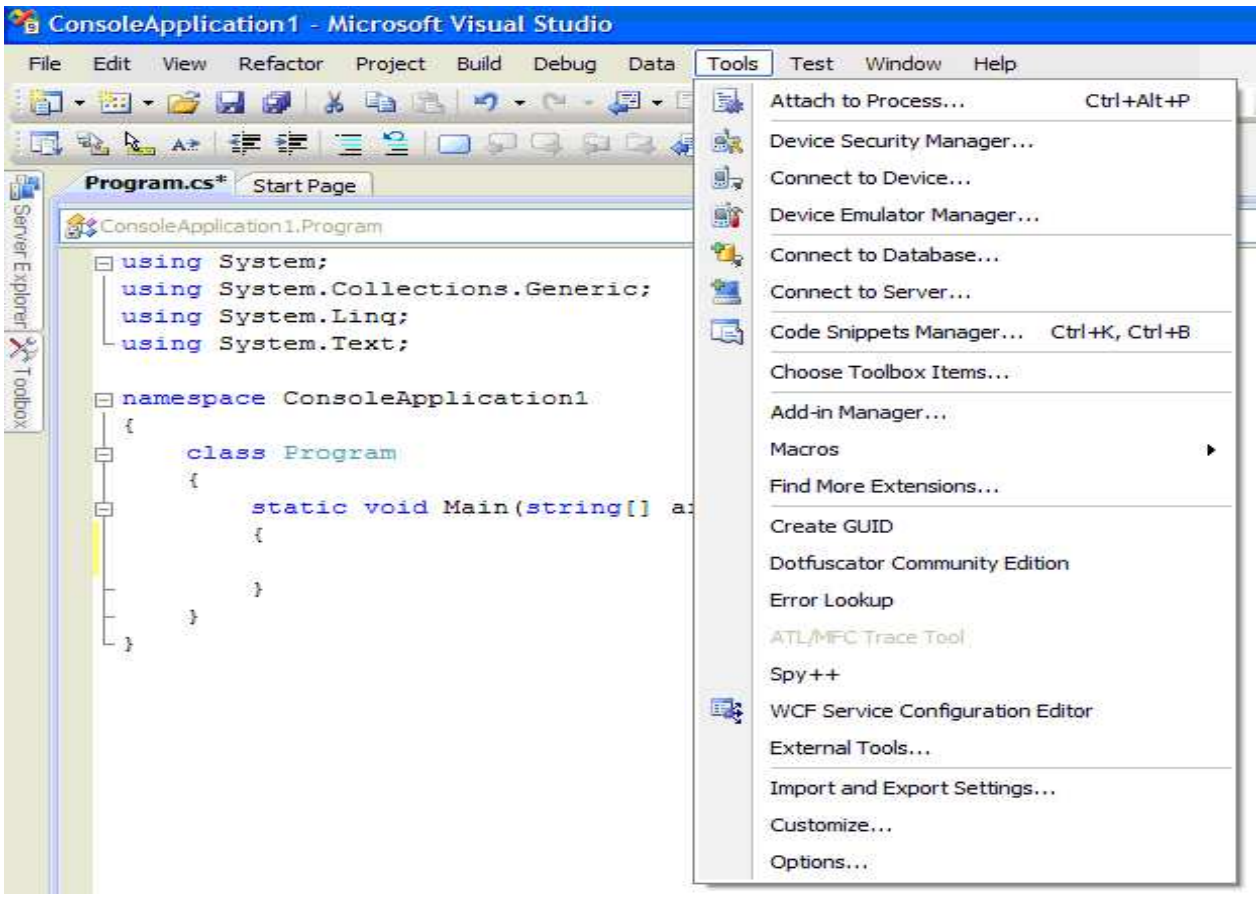


В нижнем окне видно, что компиляция прошла успешно.
Запустите программу: Debug -> Start Without Debugging:

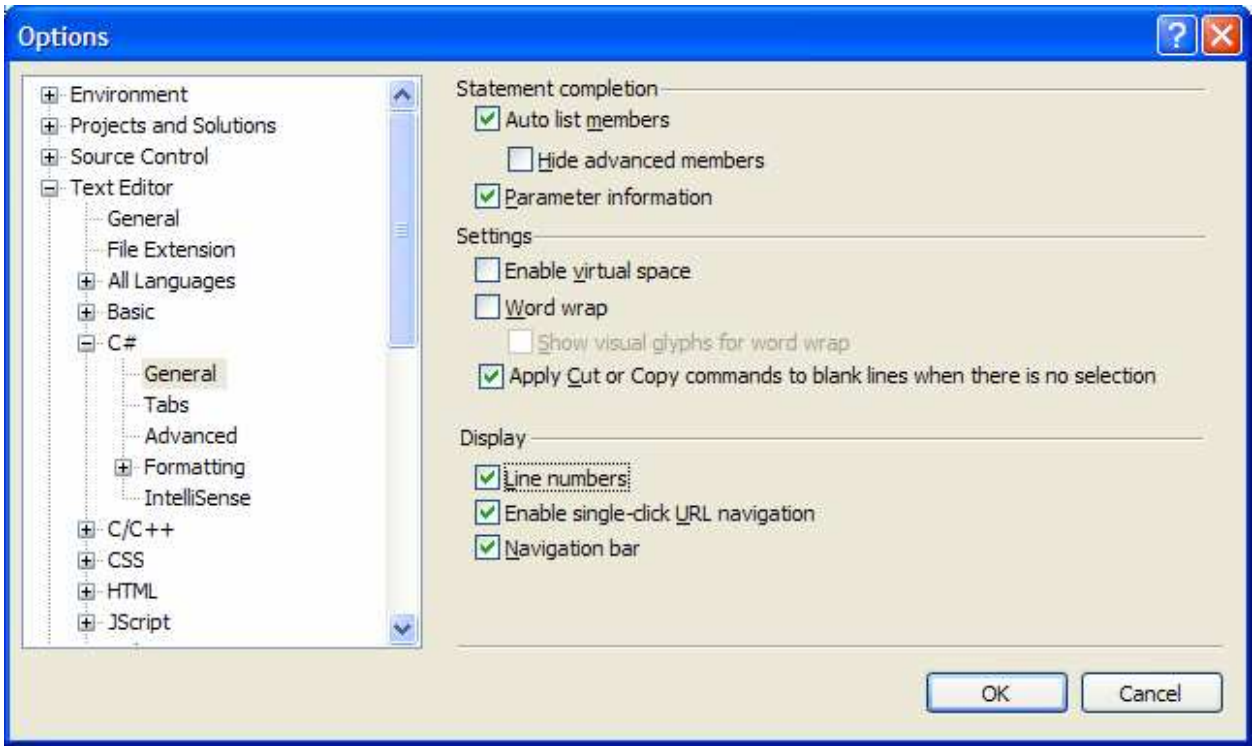


Нумерация Строк Кода в Visual Studio.

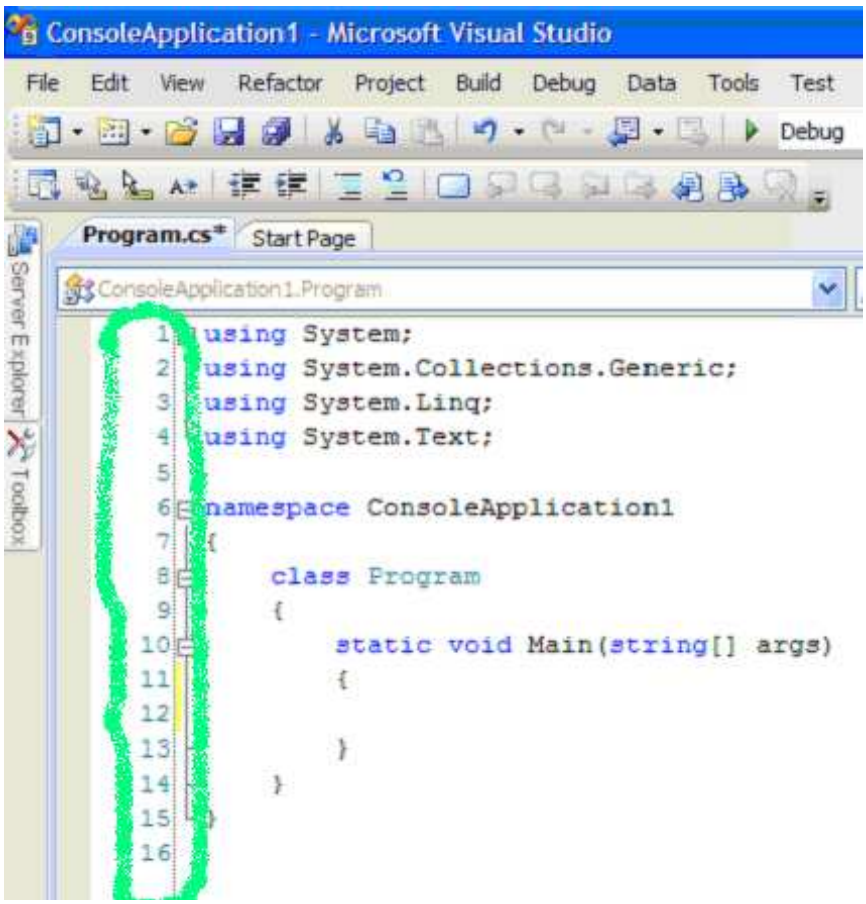
Для удобства анализа составленной программы воспользуемся свойством Visual Studio «Нумерация Строк Кода». Включить это свойство можно, пройдя по следующим окнам: пункт меню Tools –> Options. В результате будет вызвано окно:



В этом окне слева раскрывается свойство Text Editor, в свойстве C# выбирается вкладка General. В правой половине окна в поле Display отмечается галочкой свойство Line Numbers.



После нажатия клавиши Enter вид программы изменяется, появляется нумерация строк:



3 способ выполнения cs-программы.

1. Запустить интегрированную среду:
Пуск->Программы-> Microsoft Visual Studio.NET -> Microsoft Visual Studio.NET

2. Закрыть окно стартовой страницы **Start Page** (если это окно появится)

3. Создать пустой проект:
File -> New -> Project
В окне **Project types** выбрать проект **Visual C# Project**, а в окне **Project Templates** выбрать тип проекта – пустой проект **Empty Project,** после чего в окне **Name** указать имя проекта (в данном случае **Example**), а в окне **Location** - папку для хранения файлов проекта
Будет создана папка**,** в которой будет размещен файл конфигурации проекта **X.csproj,** а в окне **Solution Explorer** будет отображена структура проекта.
Примечание. При указании папки для хранения файлов проекта можно пользоваться средствами просмотра файловой структуры **Browse**

4.Подключить к проекту исходный модуль
Возможны два основных варианта: создание нового исходного модуля или подключение к проекту существующего исходного модуля
Вариант 1 (создание нового исходного модуля):
Project -> Add New Item
В окне **Categories** выбрать тип проекта **Local Project Items** , а в окне **Templates** выбрать тип файла – **Code file,** после чего в окне **Name** указать имя исходного модуля, а в окне **Location** - папку для хранения исходного модуля.

Окно редактирования будет активизировано и подготовлено для набора исходного текста программы.

Примечание. При указании папки для хранения исходного модуля можно пользоваться средствами просмотра файловой структуры **Browse**

Вариант 2 (подключение существующего исходного модуля):

Project -> Add Existing Item

В стандартном диалоговом окне выбора файлов выбрать требуемый файл с исходным кодом программы и открыть его. Окно редактирования будет активизировано и подготовлено для редактирования загруженного в него файла.

4 способ: Открытие существующего проекта

Возможны два основных варианта: запуск интегрированной среды с последующим открытием проекта или запуск проекта с автоматическим запуском интегрированной среды.

Вариант 1 :

Запустить интегрированную среду:

Пуск->Программы-> Microsoft Visual Studio.NET -> Microsoft Visual Studio.NET

Открыть существующий проект:

File -> Open -> Project

В стандартном диалоговом окне выбора файлов выбрать файл конфигурации проекта (в данном случае **X.csproj**) и открыть его. Окно редактирования будет активизировано и в него будет загружен исходный модуль проекта (в данном случае **X.cs**).

Вариант 2 :

Найти файл конфигурации проекта (в данном случае **X.csproj**) и запустить его. Автоматически будет запущена интегрированная среда, в ней будет открыт проект **X,** а в окно редактирования будет загружен исходный модуль проекта (в данном случае **X.cs**).

Запуск приложения.

После того, как одним из 4 способов программа загружена в интегрированную среду, программу надо запустить на выполнение. Запуск приложения можно осуществить в автоматическом или в отладочном режиме.

Запуск приложения в автоматическом режиме

1. Запустить интегрированную среду и открыть проект одним из рассмотренных способов.

2. Выполнить запуск проекта без отладки **Debug -> Start Without Debugging**

Примечание. В этом режиме при выполнении приложения игнорируются все установленные средства выполнения программы в отладочном режиме.

Запуск приложения в отладочном режиме

На этапе тестирования решается задача выявления смысловых ошибок в программе. Если факт наличия таких ошибок установлен, выполняется отладка программы, целью которой является определение места, характера и причины ошибки и ее устранение.

Основными методами отладки являются:

- проверка при заданных исходных данных совпадения реальной последовательности

выполнения операторов с эталонной последовательностью (*трассировка программы*);

- проверка совпадения реальных значений переменных в заданных точках программы с эталонными значениями (*контроль значений*).

На практике используется комбинация обеих методов.

При отсутствии средств отладки трассировка и контроль значений выполняются включением в исходный код программы дополнительных операторов, которые выводят значения интересующих переменных в тех точках программы, где расположены отладочные операторы. Последовательность срабатывания отладочных операторов дает информацию о трассе выполнения программы, а выводимые ими значения позволяют решать задачу контроля значений переменных.

Такой подход к отладке имеет два основных недостатка:

- Отлаживаемая и исходная программы не идентичны. Это может привести к изменению места проявления ошибки и даже ее характера;

- Сами отладочные операторы могут служить источником дополнительных смысловых ошибок.

Для устранения отмеченных недостатков в интегрированную среду разработки Microsoft Visual Studio.NET включены средства, позволяющие выполнять трассировку программы и контроль значений переменных *без внесения изменений в текст программы.*

По характеру использования эти средства можно разделить на две группы:

- средства интерактивной отладки;

- средства планируемой отладки.

Средства первой группы позволяют программисту, находясь в определенной точке программы, спланировать только один следующий шаг отладки.

Средства второй группы позволяют спланировать определенный сценарий процесса отладки на множестве шагов.

Средства интерактивной отладки позволяют реализовать гибкий сценарий отладки, однако при необходимости выполнить повторный отладочный прогон программы все отладочные действия придется выполнять заново.

Средства планируемой отладки хорошо приспособлены к повторным отладочным прогонам, но имеют меньшую гибкость отладочных действий.

В любом случае следует помнить, что *трассировка* выполняется с точностью *до одной строки* исходной программы, а *контроль значений* – с точностью *до одной переменной*. Поэтому рекомендуется не располагать в одной строке программы несколько операторов и не использовать слишком сложных выражений.

Управление средствами отладки выполняется “горячими клавишами” или через главное меню среды разработки. Далее вариант управления через “горячие клавиши” будет указываться первым, а через главное меню – вторым.

Консоль. Ввод, вывод, преобразования.

Структура C#-программы.

Программа на C# состоит из классов, внутри которых описывают методы и данные. Переменные, описанные непосредственно внутри класса, называются *полями класса*. Им автоматически присваивается так называемое «значение по умолчанию» — как правило, это 0 соответствующего типа. Переменные, описанные внутри метода класса, называются *локальными переменными*. Их инициализация возлагается на программиста.

Так называемая *область действия* переменной, то есть область программы, где можно использовать переменную, начинается в точке ее описания и длится до конца блока, внутри которого она описана. *Блок* — это код, заключенный в фигурные скобки. Основное назначение блока — группировка операторов.

В C# любая переменная описана внутри какого-либо блока: класса, метода или блока внутри метода. Имя переменной должно быть уникальным в области ее действия. Область действия распространяется на вложенные в метод блоки.

```
Пример построения программы:
// начало описания класса X
class X
{
   int A;      // поле A класса X
   int B;      // поле B класса X

//------------------------------метод Y класса X (начало)
   void Y()
   {
      int C;      // локальная переменная C, область действия - метод Y
      int A;      // локальная переменная A (НЕ конфликтует с полем A)

                           // ============ вложенный блок 1 ===========
      {
   int D;      // локальная переменная D, область действия - этот блок
//   int A;  недопустимо! Ошибка компиляции, конфликт с локальной переменной A
         C = B;      // присваивание переменной C поля B класса X (**)
         C = this.A; // присваивание переменной C поля A класса X (***)
       }             // ============ конец блока 1 ===============

                     // ============ вложенный блок 2 ============
      {
       int D;   // локальная переменная D, область действия - этот блок
      }           // ============ конец блока 2 ===============

   }   // ------------------------- конец описания метода Y

}          // конец описания класса X
```

В листинге 1 приведен пример программы, в которой описываются и выводятся на экран локальные переменные.

Листинг 1. Описание переменных

```
using System;
namespace ConsoleApplication1
{    class Class1
    {    static void Main()
        {
            int     i = 3;
            double  y = 4.12;
            decimal d = 600m;
            string  s = "Вася";

            Console.Write( "i = " );  Console.WriteLine( i );
            Console.Write( "y = " );  Console.WriteLine( y );
            Console.Write( "d = " );  Console.WriteLine( d );
            Console.Write( "s = " );  Console.WriteLine( s );
        }
    }
}
```

ВНИМАНИЕ

Переменные создаются при входе в их область действия (блок) и уничтожаются при выходе. Это означает, что после выхода из блока значение переменной не сохраняется. При повторном входе в этот же блок переменная создается заново.

Преобразования встроенных арифметических типов-значений

1. Ввод числа с плавающей точкой:

```
str = Console.ReadLine();
x = float.Parse(str);
```

2. Вывод в виде строки составного формата:

```
str = "F(" + x + ")=" + y;
Console.WriteLine(str);
```

3. Организация повторного выполнения программы:

```
Console.Write("Для повтора вычислений намите клавишу Y: ");
rep = char.Parse(Console.ReadLine());
if (rep == 'Y' || rep == 'y') goto REPEAT;
```

4. Ввод символа и представление его в виде числа двойной точности:

```
Console.Clear();
Console.Write("Введите аргумент функции:  ");
str = Console.ReadLine();
x = double.Parse(str);
```

5. Ввод исходных данных двойной точности:

```
Console.Write("Введите Y:  ");
str = Console.ReadLine();
y = double.Parse(str);
```

6. Подготовка вычислений:

```
Console.Clear();
Console.Write("Введите X:  ");
str = Console.ReadLine();
x = double.Parse(str);
Console.Write("Введите Y:  ");
str = Console.ReadLine();
y = double.Parse(str);
```

7. Ввод в программу исходных данных:

```
Console.Clear();
Console.Write("Введите первый код(цифры):  ");
str = Console.ReadLine();
cifra = int.Parse(str);
Console.Write("Введите второй код(буквы):  ");
str = Console.ReadLine();
bukva = int.Parse(str);
newbuk = bukva;
```

Линейные программы

Линейной называется программа, все операторы которой выполняются последовательно в том порядке, в котором они записаны.

Пример программы, демонстрирующей методы вывода:

```
using System;
namespace ConsoleApplication1
{    class Class1
    {    static void Main()
        {
            int     i = 3;
            double  y = 4.12;
            decimal d = 600m;
            string  s = "Вася";

            Console.WriteLine( "i = " + i );                    // 1
            Console.WriteLine( "y = {0} \nd = {1}", y, d );     // 2
            Console.WriteLine( "s = " + s );                    // 3
        }
```

```
        }
    }
```

Результат работы программы:

```
i = 3
y = 4,12
d = 600
s = Вася
```

До сих пор мы использовали метод `WriteLine` для вывода значений переменных и литералов различных встроенных типов. Это возможно благодаря тому, что в классе `Console` существует несколько вариантов методов с именами `Write` и `WriteLine`, предназначенных для вывода значений различных типов.

Методы с одинаковыми именами, но разными параметрами называются *перегруженными*. Компилятор определяет, какой из методов вызван, по типу передаваемых в него величин. Методы вывода в классе `Console` перегружены для всех встроенных типов данных, кроме того, предусмотрены варианты форматного вывода.

Листинг содержит два наиболее употребительных варианта вызова методов вывода. Если метод `WriteLine` вызван с *одним* параметром, он может быть любого встроенного типа. Нам же требуется вывести в каждой строке не одну, а две величины, поэтому прежде чем передавать их для вывода, их требуется «склеить» в одну строку с помощью операции `+`.

Перед объединением строки с числом надо преобразовать число из его внутренней формы представления в последовательность символов, то есть в строку. *Преобразование в строку* определено во всех стандартных классах C# — для этого служит метод `ToString()`. В данном случае он выполняется неявно.

Оператор 2 иллюстрирует форматный вывод. В этом случае используется другой вариант метода `WriteLine`. Первым параметром методу передается строковый литерал, содержащий, помимо обычных символов, предназначенных для вывода на консоль, *параметры* в фигурных скобках, а также *управляющие последовательности*. Параметры нумеруются с нуля, перед выводом они заменяются значениями соответствующих переменных в списке вывода.

Из управляющих последовательностей чаще всего используются символы перевода строки (`\n`) и горизонтальной табуляции (`\t`).

Рассмотрим простейшие способы *ввода с клавиатуры*. В классе `Console` определены методы ввода строки и отдельного символа, но нет методов, которые позволяют непосредственно считывать с клавиатуры числа. Ввод числовых данных выполняется в два этапа:

- Символы, представляющие собой число, вводятся с клавиатуры в строковую переменную.
- Выполняется преобразование из строки в переменную соответствующего типа.

Преобразование можно выполнить либо с помощью специального класса `Convert`, определенного в пространстве имен `System`, либо с помощью метода `Parse`, имеющегося в каждом стандартном арифметическом классе. В листинге 3.3 используются оба способа.

Пример программы, демонстрирующей методы ввода:

```
using System;
namespace ConsoleApplication1
{    class Class1
    {    static void Main()
        {
            Console.WriteLine( "Введите строку" );
            string s = Console.ReadLine();                  // 1
            Console.WriteLine( "s = " + s );
```

```
            Console.WriteLine( "Введите символ" );
            char c = (char)Console.Read();                  // 2
            Console.ReadLine();                             // 3
            Console.WriteLine( "c = " + c );

            string buf;        // строка - буфер для ввода чисел

            Console.WriteLine( "Введите целое число" );
            buf = Console.ReadLine();
            int i = Convert.ToInt32( buf );                 // 4
            Console.WriteLine( i );

            Console.WriteLine( "Введите вещественное число" );
            buf = Console.ReadLine();
            double x = Convert.ToDouble( buf );             // 5
            Console.WriteLine( x );

            Console.WriteLine( "Введите вещественное число" );
            buf = Console.ReadLine();
            double y = double.Parse( buf );                 // 6
            Console.WriteLine( y );

            Console.WriteLine( "Введите вещественное число" );
            buf = Console.ReadLine();
            decimal z = decimal.Parse( buf );               // 7
            Console.WriteLine( z );
        }
    }
}
```

Ввод строки выполняется в операторе 1. Длина строки не ограничена, ввод выполняется до символа перевода строки.

Ввод символа выполняется с помощью метода `Read`, который считывает один символ из входного потока (оператор 2). Метод возвращает значение типа `int`, представляющее собой код символа, или –1, если символов во входном потоке нет (например, пользователь нажал клавишу `Enter`). Поскольку нам требуется не `int`, а `char`, а неявного преобразования от `int` к `char` не существует, приходится применить операцию явного преобразования типа.

Оператор 3 считывает остаток строки и никуда его не передает. Это необходимо потому, что ввод данных выполняется через *буфер* — специальную область оперативной памяти. Фактически данные сначала заносятся в буфер, а затем считываются оттуда процедурами ввода. Занесение в буфер выполняется по нажатию клавиши `Enter` вместе с ее кодом. Метод `Read`, в отличие от `ReadLine`, не очищает буфер, поэтому следующий после него ввод будет выполняться с того места, на котором закончился предыдущий.

В операторах 4 и 5 используются методы класса `Convert`, в операторах 6 и 7 — методы `Parse` классов `Double` и `Decimal` библиотеки .NET, которые используются здесь через имена типов C# `double` и `decimal`.

ВНИМАНИЕ

При вводе вещественных чисел дробная часть отделяется от целой с помощью запятой, а не точки.

Если вводимые с клавиатуры символы нельзя интерпретировать как вещественное число, генерируется исключение.

Console - класс

В MSDN класс Console находится по адресу: http://msdn.microsoft.com/ru-

ru/library/system.console.aspx

Класс предоставляет стандартные потоки для консольных приложений:

- входной,
- выходной
- поток сообщений об ошибках.

Данный класс не может наследоваться.

Синтаксис: public static class Console

Организация ввода и вывода данных.

Организация ввода и вывода данных наиболее полно освещается в MSDN в виде раздела «Файловый и потоковый ввод-вывод». Доступ к этому разделу в MSDN: MSDN Library -> Разработка на .NET -> .NET Framework 4.5 -> Руководство по разработке для .NET Framework -> Основные сведения о приложениях -> Файловый и потоковый ввод-вывод.

Файловый и потоковый ввод-вывод

Пространство имен System.IO содержит типы, позволяющие выполнять синхронное и асинхронное чтение и запись данных в потоки или файлы.

Файл — это именованная и упорядоченная коллекция отдельных последовательностей байтов, имеющих постоянное место хранения. Следовательно, при работе с файлами мы оперируем такими понятиями, как путь к каталогу, дисковый накопитель, имя файла и имя каталога.

В отличие от файлов потоки предоставляют возможность писать и читать байты из вспомогательного запоминающего устройства, которым может являться одно из нескольких устройств хранения информации. Существуют как вспомогательные запоминающие устройства, отличные от дисков, так и несколько типов потоков, отличных от файловых потоков.Например:

- сетевой поток,
- поток памяти,
- поток для чтения и записи на магнитную ленту.

Распространенные задачи ввода-вывода

Пространство имен System.IO предоставляет несколько классов, которые позволяют выполнять с файлами, каталогами и потоками различные действия, такие как чтение и запись.

Распространенные задачи с файлами

Действие	Раздел с примером
Создание текстового файла.	System.IO.File
Запись в текстовый файл.	Практическое руководство. Запись текста в файл
Чтение из текстового файла.	Практическое руководство. Считывание текста из файла
	Практическое руководство. Открытие файла журнала и добавление в него данных
Добавление текста в файл.	File.AppendText

	FileInfo.AppendText
Переименование или перемещение файла.	File.Move
	FileInfo.MoveTo
Удаление файла	File.Delete
	FileInfo.Delete
Копирование файла.	File.Copy
	FileInfo.CopyTo
Получение сведений о размере файла.	FileInfo.Length
Получение атрибутов файла.	File.GetAttributes
Установка атрибутов файла.	File.SetAttributes
Определение существования файла.	File.Exists
Чтение из двоичного файла.	Практическое руководство. Считывание из нового файла данных и запись в этот файл
Запись в двоичный файл.	Практическое руководство. Считывание из нового файла данных и запись в этот файл
Извлечение расширения файла.	Path.GetExtension
Извлечение полного пути к файлу.	Path.GetFullPath
Извлечение имени и расширения файла из его пути.	Path.GetFileName
Изменение расширения файла.	Path.ChangeExtension

Распространенные задачи с каталогами

Действие	Раздел с примером
	Directory.Move
Переименование или перемещение каталога.	
	DirectoryInfo.MoveTo
Копирование каталога.	Практическое руководство. Копирование каталогов
	Directory.Delete
Удаление каталога.	
	DirectoryInfo.Delete
	Directory.CreateDirectory
Создание каталога.	
	FileInfo.Directory
Создание вложенного каталога.	DirectoryInfo.CreateSubdirectory
Просмотр файлов каталога.	FileInfo.Name
	Directory.GetDirectories
Просмотр вложенных каталогов в каталоге.	
	DirectoryInfo.GetDirectories
Отображение всех файлов во всех вложенных каталогах в указанном каталоге.	DirectoryInfo.GetFileSystemInfos
Определение размера каталога.	System.IO.Directory

Определение существования каталога. Directory.Exists

Основы файлового ввода-вывода

Абстрактный базовый класс Stream поддерживает чтение и запись байтов. Класс **Stream** поддерживает асинхронный ввод и вывод. По умолчанию его реализация определяет операции синхронного чтения и записи на основе соответствующих им асинхронных методов, и наоборот.

Все классы, которые работают с потоками, являются производными от класса **Stream**. Класс **Stream** и его производные классы предоставляют способ просмотра источников данных и хранилищ объектов, изолируя программиста конкретных от специфических деталей операционной системы и базовых устройств.

При работе с потоками используются следующие основные операции:

- Потоки могут быть считаны.Чтение — это перенос информации из потока в структуру данных, такую как массив байтов.
- В потоки можно вносить записи.Запись — это перенос информации из источника данных в поток.
- Потоки поддерживают поиск.Поиск — это выяснение и изменение текущей позиции внутри потока.

В зависимости от лежащих в основе источника или хранилища данных потоки могут поддерживать только некоторые из этих возможностей. Например, NetworkStreams не поддерживает поиск.Свойства CanRead, CanWrite и CanSeek из **Stream** и его производных классов определяют операции, поддерживаемые различными потоками.

Классы, используемые в файловом вводе и выводе

Directory предоставляет статические методы операций создания, перемещения и перечисления в директориях и поддиректориях.Класс **DirectoryInfo** предоставляет методы экземпляра.

DirectoryInfo предоставляет методы экземпляра операций создания, перемещения и перечисления в директориях и поддиректориях.Класс **Directory** предоставляет статические методы.

DriveInfo предоставляет методы экземпляра для доступа к сведениям о диске.

File предоставляет статические методы для создания, копирования, удаления, перемещения и открытия файлов, а также помогает при создании объектов FileStream.Класс **FileInfo** предоставляет методы экземпляра.

FileInfo предоставляет методы экземпляра для создания, копирования, удаления, перемещения и открытия файлов, а также помогает при создании объектов FileStream.Класс **File** предоставляет статические методы.

FileStream поддерживает произвольный доступ к файлам с помощью метода Seek.По умолчанию класс **FileStream** открывает файлы синхронно, но поддерживает и асинхронные операции.Класс **File** содержит статические методы, а класс **FileInfo** содержит методы экземпляра класса.

FileSystemInfo является абстрактным базовым классом для **FileInfo** и **DirectoryInfo**.

Path предоставляет методы и свойства для обработки строк каталогов межплатформенным способом.

DeflateStream предоставляет методы и свойства для сжатия и распаковки потоков с использованием Deflate алгоритма.

GZipStream предоставляет методы и свойства для сжатия и распаковки потоков.По умолчанию этот класс использует тот же алгоритм, что и класс DeflateStream, но он не может быть расширен для использования других форматов сжатия.

SerialPort предоставляет методы и свойства для управления файлом ресурсов порта с последовательным выводом данных.

Класс **File**, **FileInfo**, DriveInfo, **Path**, **Directory,** и **DirectoryInfo** являются изолированными (в Microsoft Visual Basic, **NotInheritable**).Можно создавать новые экземпляры этих классов, но они не могут иметь производных классов.

Классы, используемые для чтения и записи в поток

Классы BinaryReader и BinaryWriter производят чтение и запись кодированных строк и простых типов данных в **потоки**.
Класс StreamReader считывает символы из **потоков**, используя Encoding для преобразования символов в байты, и наоборот.**StreamReader** имеет конструктор, который пытается выяснить подходящую **кодировку** для заданного **потока**, на основе наличия специфичной для **кодировки**преамбулы, такой как метка порядка байтов.
StreamWriter записывает символы в **потоки**, используя **кодировку** для преобразования символов в байты.
StringReader считывает символы из **строк**.**StringReader** позволяет интерпретировать класс **Strings** с одним и тем же API-интерфейсом, поэтому в качестве выходных данных может использоваться класс **Stream** в любой кодировке, либо класс **String**.
StringWriter записывает символы в **строки**.**StringWriter** позволяет интерпретировать класс **Strings** с одним и тем же API-интерфейсом, поэтому в качестве выходных данных может использоваться класс **Stream** в любой кодировке, либо класс **String**.
TextReader является абстрактным базовым классом для класса **StringReader** и класса **StreamReader**.В то время как реализации абстрактного класса **Stream** предназначены для побайтового ввода и вывода, реализации **TextReader** предназначены для вывода знаков в кодировке Unicode.
TextWriter является абстрактным базовым классом для класса **StringWriter** и класса **StreamWriter**.В то время как реализации абстрактного класса **Stream** предназначены для побайтового ввода и вывода, реализации **TextWriter** предназначены для ввода знаков в кодировке Unicode.

Общие классы потокового ввода и вывода

Класс BufferedStream является **потоком**, который добавляет буферизацию другому **потоку**, такому как **NetworkStream**.(В классе **FileStream** буферизация является внутренним свойством, а класс **MemoryStream** не нуждается в буферизации.) Экземпляр класса **BufferedStream** может быть создан для некоторых типов потоков в целях повышения производительности ввода и вывода.Буфер — это блок байтов памяти, который используется для кэширования данных, тем самым уменьшая количество обращений к операционной системе.
Класс CryptoStream связывает потоки данных с криптографическими преобразованиями.Несмотря на то, что **CryptoStream** является производным от **Stream**, он не является частью пространства имен **System.IO**, а находится в пространстве имен System.Security.Cryptography.
MemoryStream является небуферизованным потоком, чьи инкапсулированные данные напрямую доступны в памяти.Этот поток не имеет резервного хранилища и может быть использован в качестве временного буфера.
Класс NetworkStream представляет **Поток** через сетевое подключение.Несмотря на то, что **NetworkStream** является производным от **Stream**, он не является частью пространства имен **System.IO**, а находится в пространстве имен System.Net.Sockets.

Создание списка каталогов

В следующем примере кода показано использование классов ввода/вывода для создания списка всех файлов с расширением ".exe" в каталоге.

Пример

```
using System;
using System.IO;

public class DirectoryLister
{
    public static void Main(String[] args)
    {
        string path = Environment.CurrentDirectory;
        if (args.Length > 0)
        {
            if (Directory.Exists(args[0]))
            {
                path = args[0];
            }
            else
            {
                Console.WriteLine("{0} not found; using current directory:",
                    args[0]);
            }
        }
        DirectoryInfo dir = new DirectoryInfo(path);
        foreach (FileInfo f in dir.GetFiles("*.exe"))
        {
            string name = f.Name;
            long size = f.Length;
            DateTime creationTime = f.CreationTime;
            Console.WriteLine("{0,-12:N0} {1,-20:g} {2}", size,
                creationTime, name);
        }
    }
}
```

Считывание из нового файла данных и запись в этот файл

Классы BinaryWriter и BinaryReader используются для записи и чтения данных вместо строк символов.В следующем примере кода представлены запись и чтение данных из нового пустого файлового потока (Test.data).После создания файла данных в текущем каталоге создаются соответствующие классы **BinaryWriter** и **BinaryReader**. Класс **BinaryWriter** используется для записи целых чисел от 0 до 10 в файл Test.data, при этом указатель устанавливается в конец файла.После установки файлового указателя в исходную позицию экземпляр класса **BinaryReader** считывает заданное содержимое.

Пример

```
using System;
using System.IO;
class MyStream
{
    private const string FILE_NAME = "Test.data";
    public static void Main(String[] args)
    {
        // Create the new, empty data file.  if (File.Exists(FILE_NAME))
        {
            Console.WriteLine("{0} already exists!", FILE_NAME);
            return;
        }
        FileStream fs = new FileStream(FILE_NAME, FileMode.CreateNew);
        // Create the writer for data.  BinaryWriter w = new
BinaryWriter(fs);
        // Write data to Test.data.  for (int i = 0; i < 11; i++)
        {
```

```
            w.Write( (int) i);
        }
        w.Close();
        fs.Close();
        // Create the reader for data.  fs = new FileStream(FILE_NAME,
FileMode.Open, FileAccess.Read);
        BinaryReader r = new BinaryReader(fs);
        // Read data from Test.data.  for (int i = 0; i < 11; i++)
        {
            Console.WriteLine(r.ReadInt32());
        }
        r.Close();
        fs.Close();
    }
}
```

Считывание символов из строки

С помощью приведенного ниже примера кода можно считать некоторое количество символов из существующей строки, начиная с указанной позиции в строке.Для этого, как показано ниже, используется StringReader.

Этот код определяет строку и преобразует ее в массив символов, который потом может быть считан с использованием соответствующего метода StringReader.Read.

Этот пример считывает только указанное количество символов из строки.

Пример

```
using System;
using System.IO;

public class CharsFromStr
{
    public static void Main()
    {
        // Create a string to read characters from.
        string str = "Some number of characters";
        // Make a char array the size of the source string
        char[] b = new char[str.Length];
        // Create an instance of StringReader and attach it to the string.
        StringReader sr = new StringReader(str);
        // Read 13 characters into the array that holds the string,
        // starting at the third array member.
        sr.Read(b, 0, 13);
        // Display the output.
        Console.WriteLine(b);
        // Read the rest of the string from the current position in the
        // source string into the array, starting at the 6th array member.
        sr.Read(b, 5, str.Length - 13);
        // Display the output.
        Console.WriteLine(b);
        // Close the StringReader.
        sr.Close();
    }
}
// The example has the following output:
//
// Some number o
// Some f characters
```

Запись символов в строку

С помощью приведенного ниже примера кода можно записать определенное количество символов из массива символов в существующую строку, начиная с указанной позиции в этом массиве.Для этого, как показано ниже, используется StringWriter.

Пример

```
using System;
using System.IO;
using System.Text;

public class CharsToStr
{
    public static void Main(String[] args)
    {
        // Create an instance of StringBuilder that can then be modified.
StringBuilder sb = new StringBuilder("Some number of characters");
        // Define and create an instance of a character array from which
        // characters will be read into the StringBuilder.  char[] b = {'
','t','o',' ','w','r','i','t','e',' ','t','o','.'};
        // Create an instance of StringWriter
        // and attach it to the StringBuilder.  StringWriter sw = new
StringWriter(sb);
        // Write three characters from the array into the StringBuilder.
sw.Write(b, 0, 3);
        // Display the output.  Console.WriteLine(sb);
        // Close the StringWriter.  sw.Close();
    }
}
```

Запись текста в файл

В следующих примерах демонстрируется запись текста в текстовый файл.Все текстовые файлы считываются из папки пользователя Мои документы по шаблону поиска "*.txt" и записываются в текстовый файл большого размера.
ПримечаниеПримечание

Пользователи Visual Basic могут также использовать методы и свойства, предоставляемые классом Microsoft.VisualBasic.FileIO.FileSystem для файлового ввода-вывода.
Пример

```
using System;
using System.IO;
using System.Text;
using System.Collections.Generic;

class Program
{

    static void Main(string[] args)
    {

        string mydocpath =
            Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
        StringBuilder sb = new StringBuilder();

        foreach (string txtName in
Directory.EnumerateFiles(mydocpath,"*.txt"))
        {
            using (StreamReader sr = new StreamReader(txtName))
            {
                sb.AppendLine(txtName.ToString());
```

```
                sb.AppendLine("= = = = = =");
                sb.Append(sr.ReadToEnd());
                sb.AppendLine();
                sb.AppendLine();
            }

        }

        using (StreamWriter outfile =
            new StreamWriter(mydocpath + @"\AllTxtFiles.txt"))
        {
            outfile.Write(sb.ToString());
        }
    }
}
```

Считывание текста из файла

В следующих примерах кода показаны способы чтения текста из текстового файла.Во втором примере программа выдает сообщение об обнаружении конца файла.Эту функциональную возможность можно получить с помощью метода ReadAllLines или метода ReadAllText.
Пример

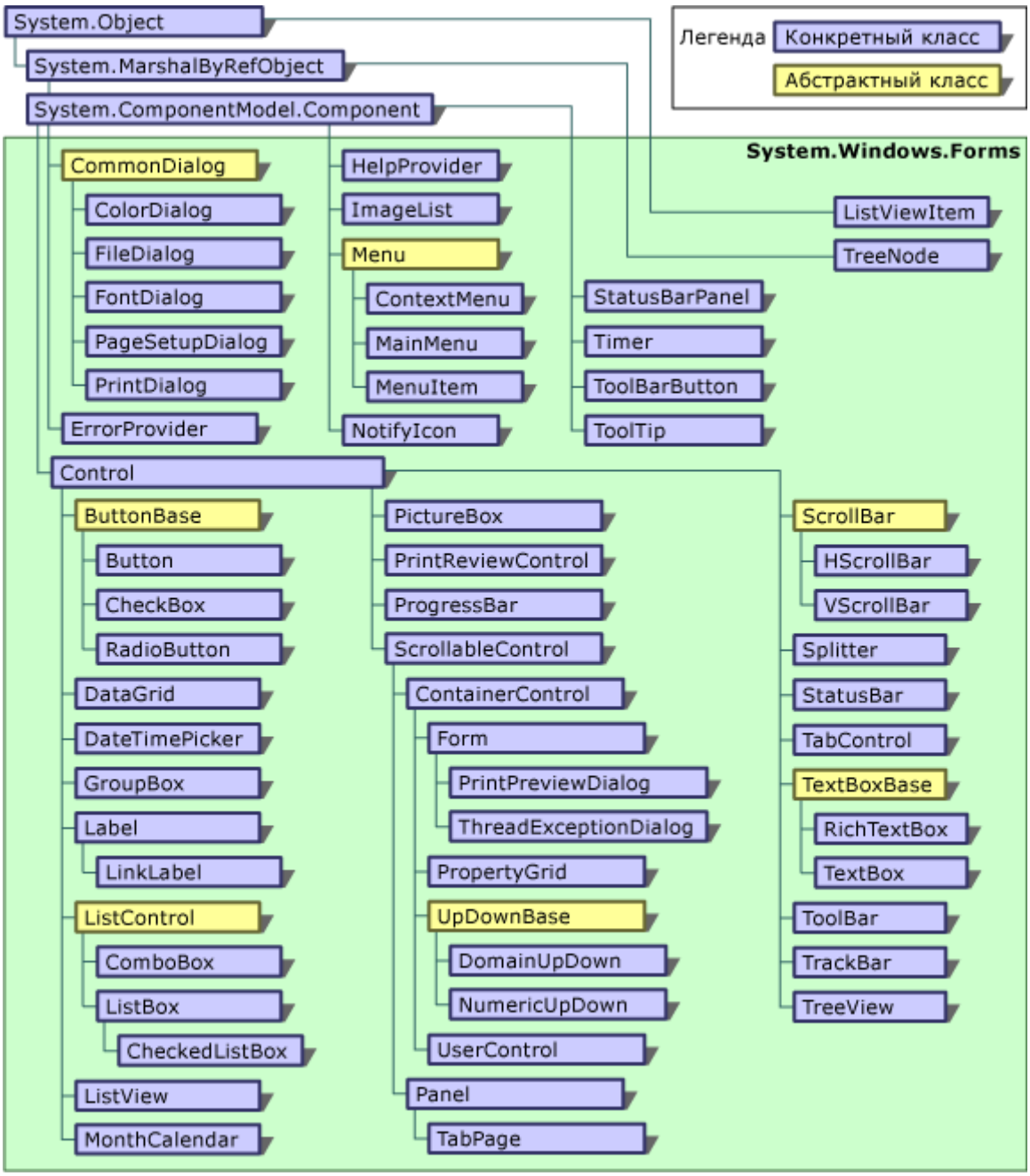
```
using System;
using System.IO;

class Test
{
    public static void Main()
    {
        try
        {
            // Create an instance of StreamReader to read from a file.// The
using statement also closes the StreamReader.using (StreamReader sr = new
StreamReader("TestFile.txt"))
            {
                String line;
                // Read and display lines from the file until the end of
                // the file is reached.while ((line = sr.ReadLine()) != null)
                {
                    Console.WriteLine(line);
                }
            }
        }
        catch (Exception e)
        {
            // Let the user know what went wrong.Console.WriteLine("The file
could not be read:");
            Console.WriteLine(e.Message);
        }
    }
}

using System;
using System.IO;
public class TextFromFile
{
    private const string FILE_NAME = "MyFile.txt";
    public static void Main(String[] args)
    {
        if (!File.Exists(FILE_NAME))
```

```
        {
            Console.WriteLine("{0} does not exist.", FILE_NAME);
            return;
        }
        using (StreamReader sr = File.OpenText(FILE_NAME))
        {
            String input;
            while ((input=sr.ReadLine())!=null)
            {
                Console.WriteLine(input);
            }
            Console.WriteLine ("The end of the stream has been reached.");
        }
    }
```

Компоновка экрана. Работа с асинхронными программами.

Асинхронный интерфейс.

Разработка на C# программ с асинхронным интерфейсом предусматривает создание такой конструкции, как формы и нанесение на неё компонент и элементов управления. Для форм Windows Forms разработано большое количество компонент. Представление об их составе и иерархии можно получить из следующей схемы, представленной в MSDN:



Стандартный состав типов компонент, необходимый для составления программы включает в себя:

1) Текст
2) Графика
3) Звук
4) Анимация
5) Инструменты: кнопка, клавиша, мышь, таймер, генератор случайных чисел, выключатель, переключатель, триггер, меню, help, указатель папки, … файла, …

Отобрать необходимые компоненты можно на основе следующей таблицы:

Функции элементов управления Windows Forms

Функция	Элемент управления	Описание
Редактирование текста	TextBox	Отображает текст, введенный во время разработки, который может

		редактироваться пользователями во время выполнения, а также может быть изменен программными средствами.
	RichTextBox	Позволяет представлять текст в простом текстовом формате или в формате RTF.
Отображение текста только для чтения	Label	Отображает текст, недоступный для непосредственного редактирования пользователем.
	LinkLabel	Отображает текст как веб-ссылку и вызывает событие, когда пользователь щелкает этот текст. Обычно такой текст является ссылкой на другое окно или на веб-узел.
	StatusBar	Отображает сведения о текущем состоянии приложения в окне, заключенном в рамку, обычно в нижней части родительской формы.
Выбор из списка	CheckedListBox	Отображает список с полосой прокрутки, состоящий из элементов с флажками.
	ComboBox	Отображает раскрывающийся список.
	DomainUpDown	Отображает список текстовых элементов, который можно прокручивать с помощью кнопок со стрелками.
	ListBox	Отображает список текстовых и графических элементов (значков).
	ListView	Отображает элементы в одном из четырех представлений: только текст, текст с маленькими значками, текст с большими значками и подробности.
	NumericUpDown	Отображает список чисел, который можно прокручивать с помощью кнопок со стрелками.
	TreeView	Отображает иерархическую структуру объектов с узлами, которые кроме текста могут включать флажки и значки.
Вывод графики	PictureBox	Отображает в рамке графические файлы, например точечные рисунки или значки.
Хранение графики	ImageList	Служит местом хранения изображений. Элементы управления ImageList и хранящиеся в них рисунки могут повторно использоваться в других приложениях.
Задание значений	CheckBox	Отображает флажок и надпись для текста. В основном используется для задания параметров.

	CheckedListBox	Отображает список с полосой прокрутки, состоящий из элементов с флажками.
	RadioButton	Выводит кнопку, которая может быть включена или выключена.
	Trackbar	Позволяет задавать значения на шкале, перемещая по ней ползунок.
Установка даты	DateTimePicker	Выводит графический календарь, позволяющий пользователю выбрать дату или время.
	MonthCalendar	Выводит графический календарь, позволяющий пользователю выбрать диапазон дат.
Диалоговые окна	ColorDialog	Отображает диалоговое окно выбора цвета, позволяющее задать цвет элемента интерфейса.
	FontDialog	Отображает диалоговое окно для задания шрифта и его атрибутов.
	OpenFileDialog	Отображает диалоговое окно для поиска и выбора файла.
	PrintDialog	Отображает диалоговое окно для выбора принтера и задания его атрибутов.
	PrintPreviewDialog	Отображает диалоговое окно, показывающее, как будет выглядеть напечатанный объект PrintDocument.
	SaveFileDialog	Отображает диалоговое окно для сохранения файла.
Элементы управления меню	MainMenu	Интерфейс режима разработки для создания меню.
	ContextMenu	Реализует меню, появляющееся при щелчке объекта правой кнопкой мыши.
Команды	Button	Используется для запуска, остановки или прерывания процесса.
	LinkLabel	Отображает текст как веб-ссылку и вызывает событие, когда пользователь щелкает этот текст. Обычно такой текст является ссылкой на другое окно или на веб-узел.
	NotifyIcon	Отображает значок в области уведомлений панели задач, соответствующий приложению, выполняемому в фоновом режиме.
	ToolBar	Содержит коллекцию набор кнопок.
Группировка других	Panel	Группирует набор элементов управления в

элементов управления		прокручиваемую рамку без надписи.
	GroupBox	Группирует набор элементов управления (например, переключателей) в непрокручиваемую рамку с надписью.
	TabControl	Страница с вкладками для эффективной организации доступа к сгруппированным объектам.

Более подробно см. Элементы управления, которые можно использовать в Windows Forms
http://msdn.microsoft.com/ru-ru/library/vstudio/x95k7xyx(v=vs.100).aspx

Создание форм без помощи Visual Studio.

На сайте http://www.firststeps.ru/dotnet/ приведено много примеров программ на языке C#. Программы приводятся с пояснениями. Вот некоторые из них, позволяющие создавать формы не через Visual Studio, а напрямую, на листке бумаги, с последующей компиляцией через консоль команд.

Windows Form

Форма это часть экрана, обычно прямоугольная, которую Вы можете использовать для того что бы предоставить пользователю информацию и получить он него ввод нужной информации.

Формы могут быть стандартным окном, **MDI-формами**, диалоговым окном и так далее.

Формы представлены классами в **Net**.

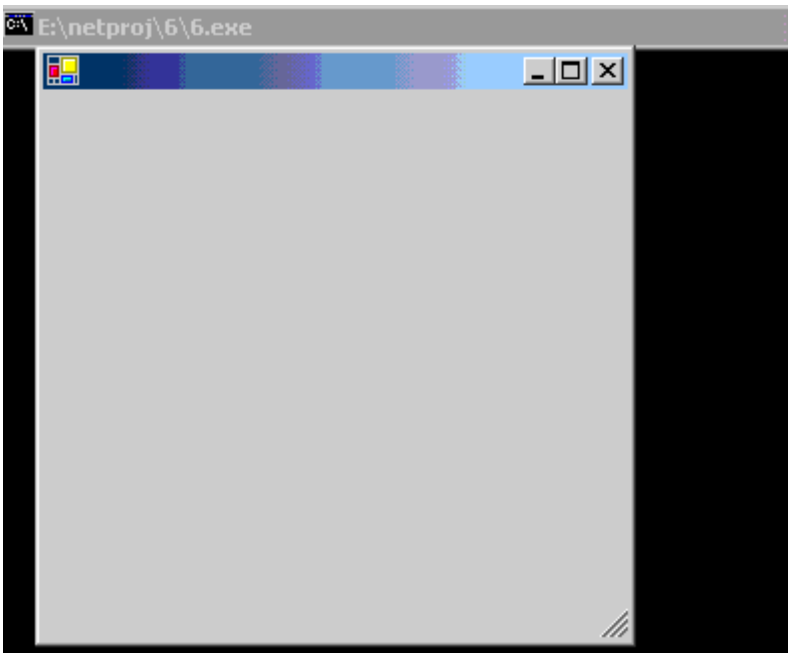
Для того что бы работать с формами нужно установить пространство имен **Windows.Form**:

```
using System.Windows.Forms;
```

После этого можно создать форму и показать ее на экране.

```
using System;
using System.Windows.Forms;

public class HelloForm
{

        public static int Main()
        {
                Form fm = new Form();
                fm.ShowDialog();
                return 0;
        }
}
```

Смотрим результат.



Идея в том, что сам класс формы, методы работы с ней находятся в самой **.Net**, а не в языке программирования и соответственно, доступны из любого языка.

Избавляемся от консольного окна

В предыдущем примере видно, что у нас за проектом есть консольное окно. Он него можно избавиться, используя опции компилятора:

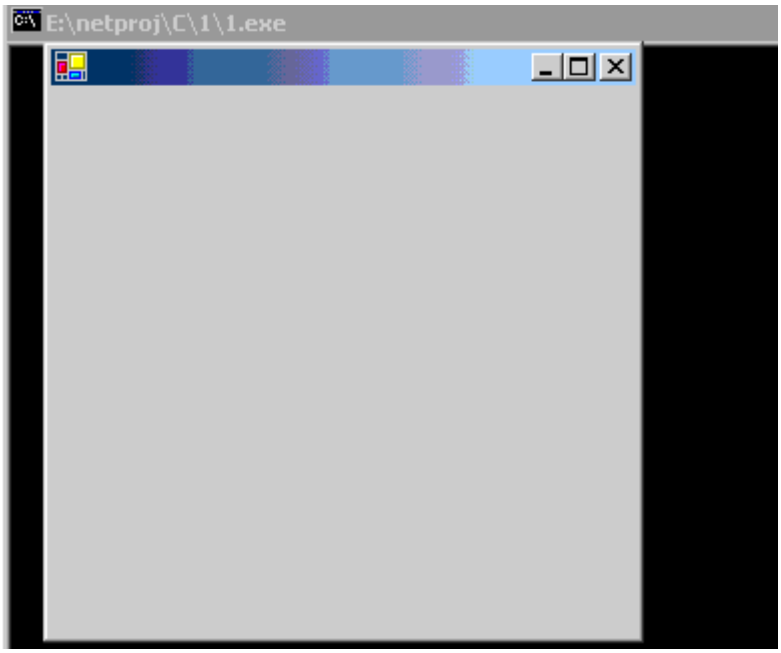
```
/target:winexe
/target:exe
```

Опция **EXE** создает консольное приложение опция **WINEXE** создает приложение на базе окна **Windows**.

Рассмотрим влияние этих опций на следующем примере. Вот код на основе **Windows.Form**.

```
using System;
using System.Windows.Forms;

class App
{
        public static void Main()
        {
                MainForm mainform = new MainForm();
                Application.Run(mainform);
        }
}

class MainForm:Form
{
}
```

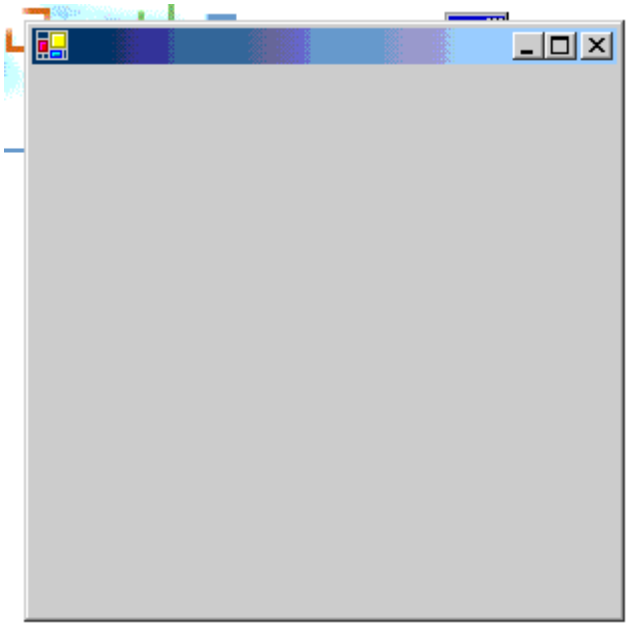
Результат работы будет такой – за формой видно окно командной консоли.



По команде:

csc.exe /target:winexe 1.cs.

окно командной консоли устраняется:



Создаем окно

Итак, наша задача научиться создавать окна. Пишем код.

```
using System;
using System.Windows.Forms;

class MyForm : Form
{
        public static void Main()
        {
```

```
                Application.Run(new MyForm());
        }
}
```

Здесь многое знакомо. Для того, что бы пользоваться окнами нужно, что бы классы окон были доступны. Для этого мы объявляем пространство имен.

```
using System.Windows.Forms;
```

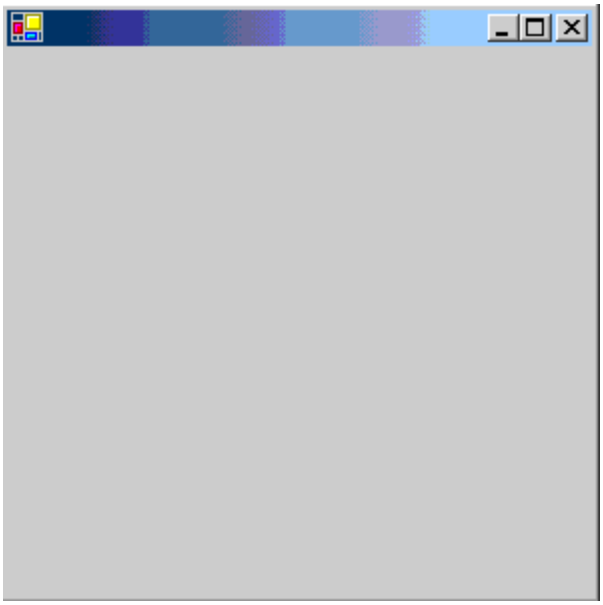
Дальше у нашей программы должен быть главный класс, класс все программы. Мы его объявляем как наследник от класса **Form**.

```
class MyForm : Form
```

Нам нужна функция Main - эта функция есть в каждой программе, вот мы ее и объявляем в классе **MyForm**.

```
class MyForm : Form
{
        public static void Main()
        {

        }
}
```

Ну и последнее связано с тем, что нам нужно запустить цикл обработки сообщений. Метод **Application.Run** запускает цикл обработки сообщений. Этот метод находиться в классе **Application** который отвечает за все приложение, за его запуск и остановку, за циклы обработки сообщений и так далее. Вот так будет выглядеть наше приложение.



Добавляем меню

Итак, окно есть давайте посмотрим, как можно добавить меню.

```
using System;
using System.Windows.Forms;

class MyForm : AppForm
```

```
{
        public static void Main()
        {
                Application.Run(new MyForm());
        }
}


class AppForm : Form
{
        public  AppForm()
        {
                MainMenu mnuFileMenu = new MainMenu();
                this.Menu = mnuFileMenu;
                mnuFileMenu.MenuItems.Add("Test");
        }
}
```

Во-первых, мы создали свой новый класс от которого будет производиться наследование. И перегрузили конструктор. В конструктору мы будем создавать меню. **MainMenu** это класс который будет на форме. У него есть ряд методов и свойств, например для добавления пунктов к меню. Но создать этот класс и связать его с окном это еще не все. Нужно связать этот класс с формой. У нас есть свойство у формы - **Menu**, с помощью которого можно связать класс меню с формой.

```
this.Menu = mnuFileMenu;
```

Смотрим результат.

Свойства (properties)

По аналогии с **COM** объектами было введено понятие свойства. Каждый **C#** класс может иметь свойства и может использоваться как **COM** объект. **C#** позволяет определять свойства внутри любого класса. Внутри **C#** класса, каждому свойству дается имя и тип данных. Ключевые слова **set** и **get** используется для объявления выполняемого кода при чтении или обновлении свойства. Идея в том, что для свойств функции ввода и получения в одном имени не надо писать две функции.

```
SetValue
GetValue
```

При этом, контроль при вводе. Давайте пробовать.

```
using System;
using System.Windows.Forms;

class MyProgramm
{
	public static void Main()
	{
		MyClass mc = new MyClass();
		mc.Text="Hello";
		Console.Write(mc.Text);
	}
}


class MyClass
{
	public string Text
	{
		get
		{
			return myString;
		}
		set
		{
			myString=value;
		}
	}
	private string myString;
}
```

Используя имя свойства можно передавать и получать данные. В прошлом примере для создания меню мы использовали свойство **Menu** передавая класс.

```
MainMenu mnuFileMenu = new MainMenu();
this.Menu = mnuFileMenu;
```

Обработка событий на форме

Давайте сразу посмотрим код.

```
using System;
using System.Windows.Forms;
using System.Drawing;

class MyForm : AppForm
{
	public static void Main()
	{
		Application.Run(new MyForm());
	}
}


class AppForm : Form
{
	public  AppForm()
```

```
        {
                MainMenu mnuFileMenu = new MainMenu();
                this.Menu = mnuFileMenu;
                mnuFileMenu.MenuItems.Add("Test");
        }

        protected override void OnMouseDown(MouseEventArgs e)
        {
                MessageBox.Show("You clicked on Form ","First Step Site");
        }
}
```

Мы здесь с Вами обработали событие нажатия клавиши мышки. У класса Form есть ряд предопределенных функций, которые вызываются при тех или иных событиях. Например для мышки есть следующие события.

```
MouseDown
MouseEnter
MouseHover
MouseLeave
MouseMove
MouseUp
MouseWheel
Move
```

Нам остается в классе переопределить это событие и установить свою реакцию на него. Вот так будет выглядеть результат нажатия левой клавиши мышки.

Изменение размера формы

Смотрим код:

```
using System.Windows.Forms;
using System.Drawing;

class MyForm : AppForm
{
        public static void Main()
        {
                Application.Run(new MyForm());
        }
}


class AppForm : Form
{
```

```
        public  AppForm()
        {
                MainMenu mnuFileMenu = new MainMenu();
                this.Menu = mnuFileMenu;
                mnuFileMenu.MenuItems.Add("Test");
        }

        protected override void OnMouseDown(MouseEventArgs e)
        {
                this.Size = new System.Drawing.Size(100, 100);
        }

        protected override void OnMouseUp(MouseEventArgs e)
        {
                this.Width = 400;
                this.Height = 400;
        }
}
```

У нас есть два способа изменять размер. Используя свойство **Size** или **Width** и **Height**. В первом способе мы воспользовались системной структурой **Size,** передав ее в качестве параметра, во втором - свойствами ширины и высоты. Теперь при нажатии клавиши мыши форма будет уменьшаться, а при отпускании увеличиваться.

Изменение положения формы

Смотрим код:

```
using System;
using System.Windows.Forms;
using System.Drawing;

class MyForm : AppForm
{
        public static void Main()
        {
                Application.Run(new MyForm());
        }
}


class AppForm : Form
{
        public  AppForm()
        {
                MainMenu mnuFileMenu = new MainMenu();
                this.Menu = mnuFileMenu;
                mnuFileMenu.MenuItems.Add("Test");
        }

        protected override void OnMouseDown(MouseEventArgs e)
        {
                this.Location = new Point (0, 0);
        }
        protected override void OnMouseUp(MouseEventArgs e)
        {
                this.Left =200 ;
                this.Top = 200;
        }
}
```

Есть у нас свойство **Location** отвечающее за положение формы, которая принимает параметр **Point** и есть два свойства **Letf** и **Top** - эти свойства дублируют **Location,** позволяя обращаться к левому и верхнему углу.

Встраиваем элемент управления в окно

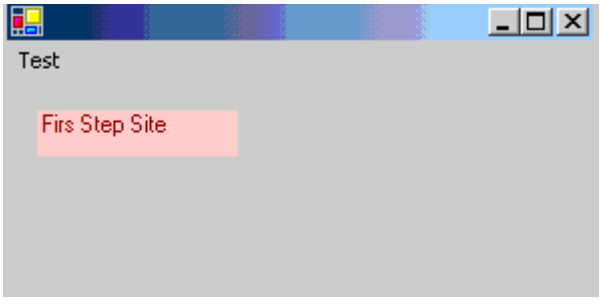
Смотрим код:

```
using System;
using System.Windows.Forms;
using System.Drawing;

class MyForm : AppForm
{
        public static void Main()
        {
                Application.Run(new MyForm());
        }
}


class AppForm : Form
{
        public  AppForm()
        {
                MainMenu mnuFileMenu = new MainMenu();
                this.Menu = mnuFileMenu;
                mnuFileMenu.MenuItems.Add("Test");
                Label label1 = new Label();
                label1.Text = "Firs Step Site";
                label1.Location = new Point(15,15);
                label1.BackColor = Color.Pink;
                label1.ForeColor = Color.Maroon;
                this.Controls.Add(label1);
        }
}
```

Итак, у нас есть некоторое количество элементов управления изначально в системе. Один из них **Label**. Мы его создаем и настраиваем свойства. Это вроде как понятно. Но вот что нового для программистов от **MFC** и не ново для программистов от **VB**. У нас есть коллекция элементов управления на окне. И мы спокойно этот элемент управления добавляем в коллекцию.

```
this.Controls.Add(label1);
```

А вот как это выглядит.



Обработка сообщений элемента классом элемента

Итак, нам нужно научиться обрабатывать сообщений от элемента управления. Мы будет делать это внутри класса элемента. То есть класс элемента будет сам реагировать на сообщения. Пишем код:

```
using System;
using System.Windows.Forms;
using System.Drawing;

class MyForm : AppForm
{
        public static void Main()
        {
                Application.Run(new MyForm());
        }
}


class AppForm : Form
{
        public  AppForm()
        {
                MainMenu mnuFileMenu = new MainMenu();
                this.Menu = mnuFileMenu;
                mnuFileMenu.MenuItems.Add("Test");
                MyLabel label1 = new MyLabel();
                label1.Text = "Firs Step Site";
                label1.Location = new Point(15,15);
                label1.BackColor = Color.Pink;
                label1.ForeColor = Color.Maroon;
                this.Controls.Add(label1);
        }
}

class MyLabel : Label
{
        protected override void OnMouseDown(MouseEventArgs e)
        {
                MessageBox.Show("You clicked on Label","First Step Site");
        }
}
```

Как видите методика простая. В самом классе **Label** есть методы которые вызываются при наступлении сообщения. Нам нужно просто их перегрузить. Мы создали свой класс как наследник от Label. Перегрузили метод OnMouseDown, а на форме создали уже элемент управления на основе нашего класса.

```
MyLabel label1 = new MyLabel();
```

Вот и все. Осталось только запустить и проверить.

Создание меню подробнее

В http://www.firststeps.ru/dotnet/r.php?19 можно посмотреть, как вообще можно создавать меню. Смотрим код:

```
using System;
using System.Windows.Forms;
using System.Drawing;

class MyForm : AppForm
{
        public static void Main()
        {
                Application.Run(new MyForm());
        }
}


class AppForm : Form
{
        public  AppForm()
        {
                MainMenu mnuFileMenu = new MainMenu();
                this.Menu = mnuFileMenu;
                MenuItem MenuItemFile = new MenuItem("&File");
                MenuItemFile.MenuItems.Add("New");
                MenuItemFile.MenuItems.Add("Open");
                MenuItemFile.MenuItems.Add("Save");
                MenuItemFile.MenuItems.Add("Exit");
                MenuItem MenuItemEdit = new MenuItem("&Edit");
                MenuItemEdit.MenuItems.Add("Copy");
                MenuItemEdit.MenuItems.Add("Paste");
                mnuFileMenu.MenuItems.Add(MenuItemFile);
                mnuFileMenu.MenuItems.Add(MenuItemEdit);
                MenuItem MenuItemEditExt = new MenuItem("From File");
                MenuItemEditExt.MenuItems.Add("In File");
                MenuItemEditExt.MenuItems.Add("To File");
                MenuItemEdit.MenuItems.Add(MenuItemEditExt);
        }
}
```

Что здесь главное. Есть такой класс как **MenuItem**. Этот класс позволяет создавать субменю. Мы субменю создали три класса.

```
MenuItem MenuItemFile = new MenuItem("&File");
MenuItem MenuItemEdit = new MenuItem("&Edit");
```

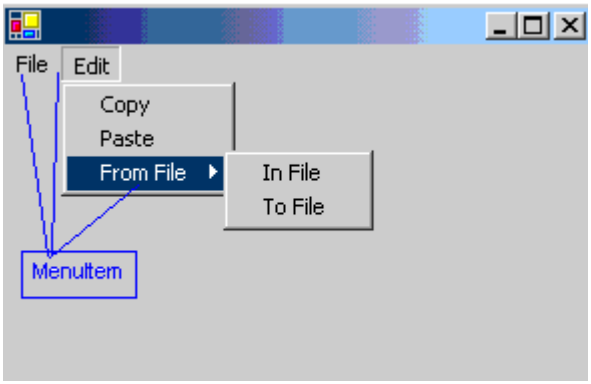
И добавили два из них к главному меню.

```
MainMenu mnuFileMenu = new MainMenu();
this.Menu = mnuFileMenu;
mnuFileMenu.MenuItems.Add(MenuItemFile);
mnuFileMenu.MenuItems.Add(MenuItemEdit);
```

А третий класс субменю мы добавили к меню **Edit**.

```
MenuItem MenuItemEditExt = new MenuItem("From File");
MenuItemEditExt.MenuItems.Add("In File");
MenuItemEditExt.MenuItems.Add("To File");
MenuItemEdit.MenuItems.Add(MenuItemEditExt);
```

Тем самым создав субменю внутри меню **Edit**. Вот так это выглядит.



Создаем обработчик событий меню

Итак, меню у нас есть, а вот как обрабатывать события этого меню? Давайте посмотрим код, который умеет это делать.

```
using System;
using System.Windows.Forms;
using System.Drawing;

class MyForm : AppForm
{
        public static void Main()
        {
                Application.Run(new MyForm());
        }
}


class AppForm : Form
{
        public  AppForm()
        {
                MainMenu mnuFileMenu = new MainMenu();
                this.Menu = mnuFileMenu;
                MenuItem MenuItemFile = new MenuItem("&File");
                MenuItem MenuNew = new MenuItem("New",new
System.EventHandler(this.MenuNew_Click));
                MenuItemFile.MenuItems.Add(MenuNew);
                MenuItemFile.MenuItems.Add("Open");
                MenuItemFile.MenuItems.Add("Save");
                MenuItemFile.MenuItems.Add("Exit");
```

```
                MenuItem MenuItemEdit = new MenuItem("&Edit");
                MenuItemEdit.MenuItems.Add("Copy");
                MenuItemEdit.MenuItems.Add("Paste");
                mnuFileMenu.MenuItems.Add(MenuItemFile);
                mnuFileMenu.MenuItems.Add(MenuItemEdit);
                MenuItem MenuItemEditExt = new MenuItem("From File");
                MenuItemEditExt.MenuItems.Add("In File");
                MenuItemEditExt.MenuItems.Add("To File");
                MenuItemEdit.MenuItems.Add(MenuItemEditExt);
        }

        private void MenuNew_Click(Object sender, EventArgs e)
        {
                MessageBox.Show("Menu New");
        }
}
```

Вся суть обработки находится в коде:

```
MenuItem MenuNew = new MenuItem("New",new
System.EventHandler(this.MenuNew_Click));
MenuItemFile.MenuItems.Add(MenuNew);
```

Мы создали класс **MenuItem** и в этом класс при создании указали не только название функции на и указали какая функция будет производить обработку сообщений от этого класса.

```
new System.EventHandler(this.MenuNew_Click));
```

Ну и позже описали саму функцию.

```
private void MenuNew_Click(Object sender, EventArgs e)
{
        MessageBox.Show("Menu New");
}
```

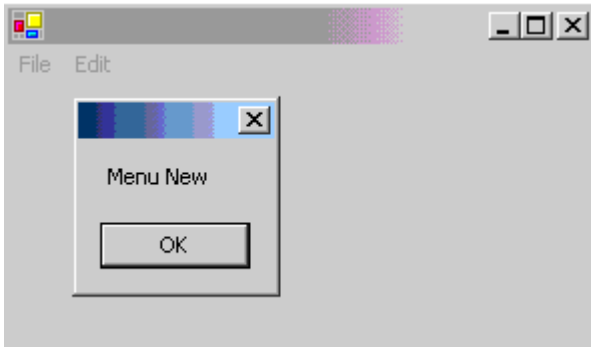
Идти по такому длинному пути совсем не обязательно, намного проще указать обработчик при добавлении пункта меню.

```
MenuItemFile.MenuItems.Add("Open",new
System.EventHandler(this.MenuOpen_Click));
```

Ну и саму функцию обработки написать.

```
private void MenuOpen_Click(Object sender, EventArgs e)
{
        MessageBox.Show("Menu Open");
}
```

Вообщем, при создании меню кроме имени нужно указывать еще и функцию обработки события. Приложение реагирует на меню.

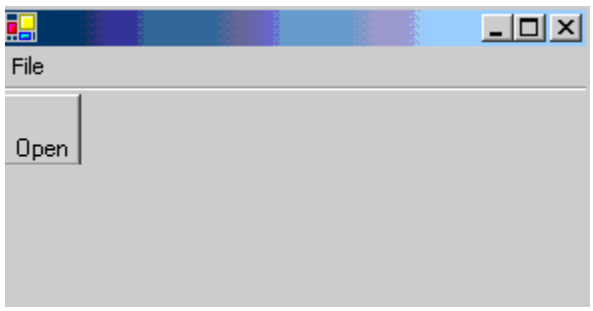


Создаем панель инструментов

Итак давайте посмотрим как создавать панель инструментов. Код:

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.IO;

class MyForm : AppForm
{
        public static void Main()
        {
                Application.Run(new MyForm());
        }
}


class AppForm : Form
{
        public AppForm()
        {
                MainMenu mnuFileMenu = new MainMenu();
                this.Menu = mnuFileMenu;
                MenuItem MenuItemFile = new MenuItem("&File");
                mnuFileMenu.MenuItems.Add(MenuItemFile);
                ToolBar toolBar1 = new ToolBar();
                ToolBarButton toolBarButton1 = new ToolBarButton();
                toolBarButton1.Text = "Open";
                toolBar1.Buttons.Add(toolBarButton1);
                Controls.Add(toolBar1);
        }
}
```

Все как всегда. Создаем объект класса **ToolBar** в него добавляем объекты классов **toolBarButton** и в конце добавляет сам класс панели инструментов к коллекции классов на форме.

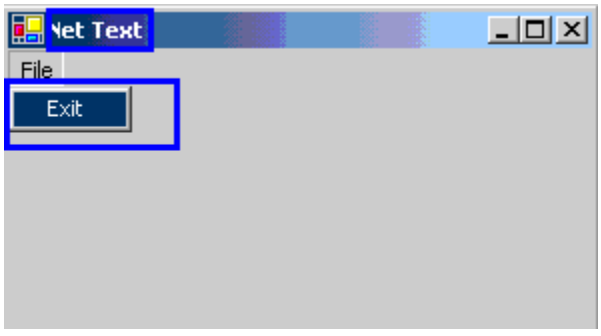


Заголовок формы и пункт меню выход

Как всегда код:

```
using System;
using System.Windows.Forms;

class MyForm : AppForm
{
        public static void Main()
        {
                Application.Run(new MyForm());
        }
}


class AppForm : Form
{
        public AppForm()
        {
                Text="Net Text";
                MainMenu mnuMenu = new MainMenu();
                this.Menu = mnuMenu;
                MenuItem MenuItemFile = new MenuItem("&File");
                MenuItem MenuExit = new MenuItem("Exit",new
System.EventHandler(this.MenuExit_Click));
                MenuItemFile.MenuItems.Add(MenuExit);
                mnuMenu.MenuItems.Add(MenuItemFile);
        }

        private void MenuExit_Click(Object sender, EventArgs e)
        {
                Application.Exit();
        }
}
```

Все довольно тривиально, есть свойство **Text** которое отвечает за заголовок. А по нажатию на **Exit** вызывается метод класса **Application** который отвечает за работу приложение и который с радостью приложение закроет.



Компонентное программирование. «Сборка проекта и работа с ним».

При сборке проекта используется компилятор csc и некоторые другие утилиты пакета Visual Studio. Основными операциями при сборке программы являются: создание и использование динамических библиотек, организация взаимодействия с различными видами ресурсов.

Создание DLL

Для создания DLL на C# нужно просто написать класс без функции Main.

```
using System;

namespace MyClass
{
	public class My
	{
		public static string MyStrimng(string s)
		{
			return s+"Hello";
		}
	}
}
```

и сохранить его в файле dll.cs.

Для компиляции его ключ /target нужен, чтобы указать, как будет компилироваться проект library.

Вот полный пример команды компиляции:

```
csc.exe /target:library /out:Mainw.dll dll.cs
```

В результате компиляции появиться файл dll.cs и библиотечный файл Mainw.dll.

Обратите внимание: класс в данном случае должен быть public, а методы которые будут доступны должны быть описаны как static.

Старайтесь в именах не пересекаться со стандартными (зарезервированными) именами. Например использование имени Main, а не Mainw для DLL грозит ошибками.

Использование DLL

Для использования DLL нужна программа-клиент, содержащая метод Main и оператором using определяющая, что для её работы будет нужно пространство имён MyClass. Такое пространство имён нами было указано в файле dll.cs при создании динамической библиотеки – Main.dll.

Код программы-клиента может содержать обращение к элементам (свойствам, методам, объектам и др.), определённым в пространстве имён MyClass:

```
using System;
using System.Windows.Forms;
using MyClass;

class MainClasses
{
	public static void Main(string[] args)
	{
		string s=My.MyStrimng("client ");
		Console.Write(s);
	}
}
```

Сохраним её в файле 1.cs.

При компиляции нужно сослаться на используемую динамическую библиотеку (DLL) - для этого в компиляторе csc есть опция /reference. Вот готовая команда компиляции:

```
csc /out:Main.exe /reference:Mainw.dll 1.cs
```

В результате появляется файл Main.exe, который будет в своей работе использовать динамическую библиотеку.

Результат работы созданной сборки из двух файлов (Main.exe и Mainw.dll):

Создание модуля и сборки (Assembly)

Если необходимо сделать сборку с другими файлами в dll, мы должны откомпилировать дополнительно подключаемый проект, как модуль. Для этого есть специальный ключ /target:module.

Команда компиляции:

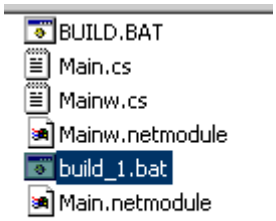
```
csc.exe /target:module  Mainw.cs
```

В результате компиляции мы получим именно модуль с расширением netmodule.

Теперь добавим модуль созданного ранее клиентского приложения:

```
csc /addmodule:Mainw.netmodule  /t:module Main.cs
```

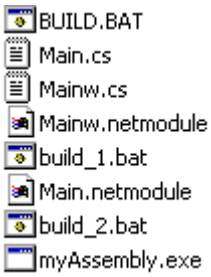
Появится новый модуль:



И вот теперь мы можем создать сборку воспользовавшись утилитой AL:

```
al Main.netmodule Mainw.netmodule /main:MainClasses.Main /out:myAssembly.exe
/target:exe
```

И вот только теперь, если Все нормально и нет ошибок вы получите сборку.



Текст созданного ранее клиентского приложения:

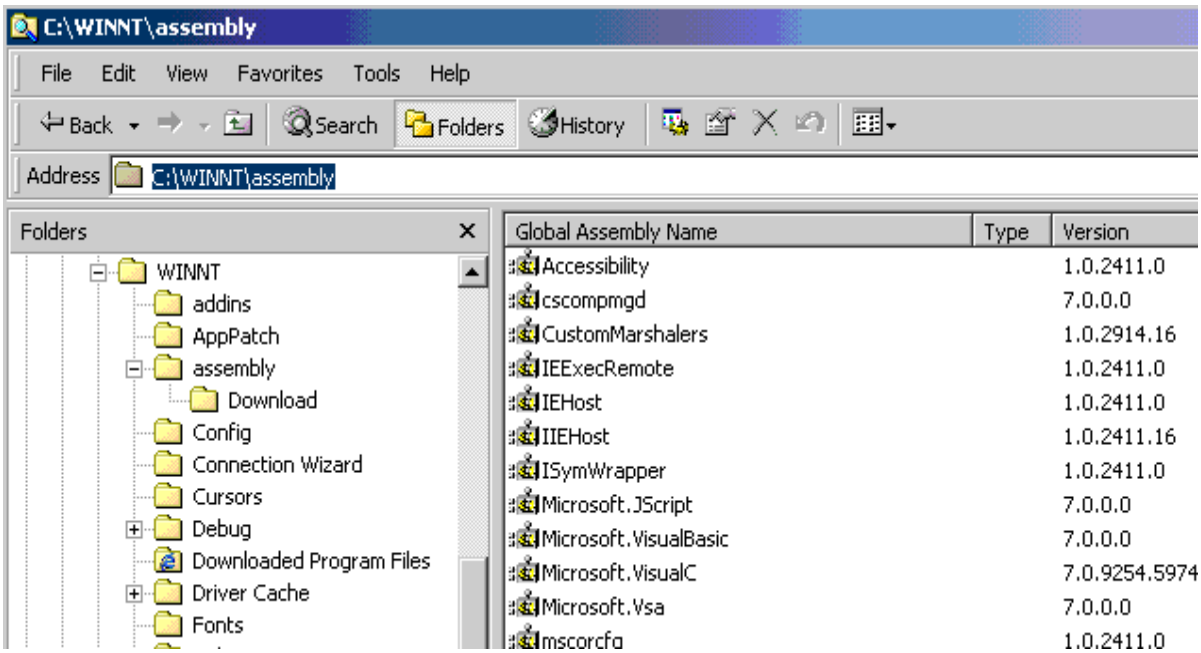
```
using System;
using System.Windows.Forms;
using MyClass;

class MainClasses
{
 public static void Main(string[] args)
 {
        string s=My.MyStrimng("client ");
        Console.Write(s);
 }
}
```

Global Assembly Cache

- это то место, в котором находятся сборки. Если сборка будет распределена между многими приложениями то она должна быть установлена в это место. Оно находится по пути:

```
:\WINNT\assembly
```

При просмотре его будет использована Shfusion.dll которая отвечает за просмотр.



Добавлять и удалять сборку из глобального кеша можно с помощью утилиты gacutil:

```
Добаляем
gacutil /i mydll.dll
Удаляем
gacutil /u mydll
```

Использование стандартных DLL

Несмотря на большое количество классов в Net Classes, все равно может потребоваться использовать старое DLL. Например, если Вы переписываете проект и у Вас есть возможность использовать старые DLL, кроме того вообще часть функций реализована в виде DLL и наверно дальше будет реализовываться.

Создание DLL на C# очень сильно упрощено что говорит о том что они будут использоваться дальше. Смотрим код.

```
using System;
using System.Runtime.InteropServices;

class MainClass
{

[DllImport("kernel32")]
public static extern bool  Beep(int _Int1, int _Int2);

 public static void Main(string[] args)
 {
        Beep(300,100);
 }
 }
```

Мы использовали DLLImport указав имя библиотеки и после нее описали функцию, которую будем использовать, описывать ее нужно с правильными параметрами а для этого посмотреть WIN32 API.

Этот код издаст звук.

Файл с ресурсами строк

http://www.firststeps.ru/dotnet/r.php?45

Файл с ресурсами строк должен иметь расширение TXT. В нем могут находиться только строки. Обычно он имеет следующий формат:

```
[header]                заголовок
;comments               комментарии
name = value    значения
```

Давайте создадим тестовый файл с именем resource.txt и текст в нем.

```
string1=Hello resouce file
string2=My Resource File
```

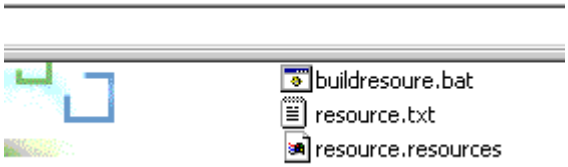
А теперь нам нужно воспользоваться утилитой resgen для получения файла ресурсов. Создадим файл с именем buildresoure.bat и код в нем.

```
Resgen.exe resource.txt
```

Все запускаем на выполнение.

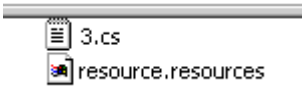
```
D:\net_step\2>buildresoure

D:\net_step\2>Resgen.exe resource.txt
Read in 2 resources from 'resource.txt'
Writing resource file...  Done.

D:\net_step\2>
```

В результате будет сформирован файл resource.resources.



Извлечение ресурсов из файла

Поместим только что созданный файл в туже самую папку, что и проект.

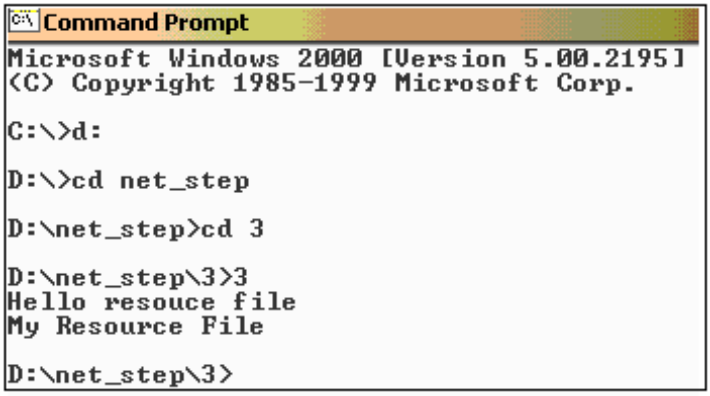


Напишем код.

```
using System;
using System.Resources;
class MainApp
{
    public static void Main()
    {
        ResourceManager rm =
```

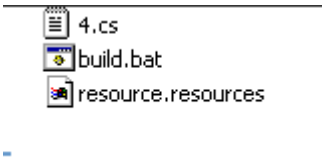
```
ResourceManager.CreateFileBasedResourceManager("resource",".",null);
        Console.WriteLine(rm.GetString("string1"));
        Console.WriteLine(rm.GetString("string2"));
    }
}
```

Для того, чтобы работать с ресурсами нам нужно пространство имен System.Resources там находятся классы для работы с ресурсами. Один из этих классов ResourceManager он отвечает за загрузку ресурсов и управление ими. Мы загрузили ресурсы из файла, и вывели строки на экран.



Присоединение файла ресурсов к исполняемому файлу (exe)

Чтобы прикрепить файл с ресурсами к программе, файл ресурсов должен быть уже скомпилирован.



Соединение файла EXE и файла ресурсов производиться при компиляции. Это делается путем использованию ключа /res при компиляции. Вот пример команды:

```
csc /res:resource.resources 4.cs
```

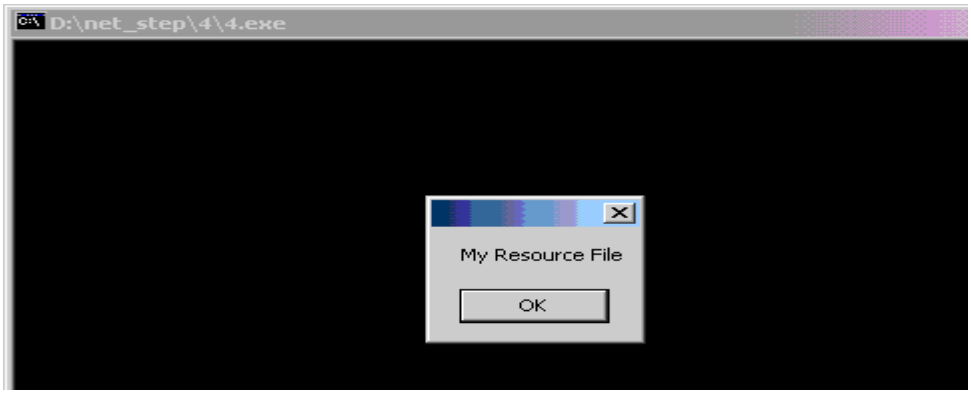
А вот код программы:

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.IO;
using System.Resources;


class MyForm : AppForm
{
        public static void Main()
        {
                Application.Run(new MyForm());
        }
```

```
}


class AppForm : Form
{
        public AppForm()
        {
                rm = new ResourceManager("resource",this.GetType().Assembly);
                MessageBox.Show(rm.GetString("string2"));
        }
        private ResourceManager rm;
}
```

Здесь мы воспользовались классом ResourceManager который умеет работать с ресурсами, загружать их, получать ресурс по названию. Он находиться в пространстве имен using System.Resources;. Воспользовались функций этого класса GetString для получения строки из ресурсов.

Итак компилируем и запускаем. У нас появиться окно перед запуском формы в котором будет строка из файла ресурса.



Первоначальный текстовый файл был такой:

```
string1=Hello resouce file
string2=My Resource File
```

Дополнительная информация:

1. Кроме того, в firststeps.ru имеются и другие примеры по работе с ресурсами:

- "Шаг 44 - Извлечение ресурсов из модуля EnumResourceTypes()"
- "Шаг 47 - Поиск ресурсов FindResource"
- "Шаг 48 - LoadIcon()"

Шаг 181 - Подробнее о ресурсах

http://www.firststeps.ru/mfc/steps/r.php?181

Шаг 310 - Почему ресурсы бывают под одинаковыми именами

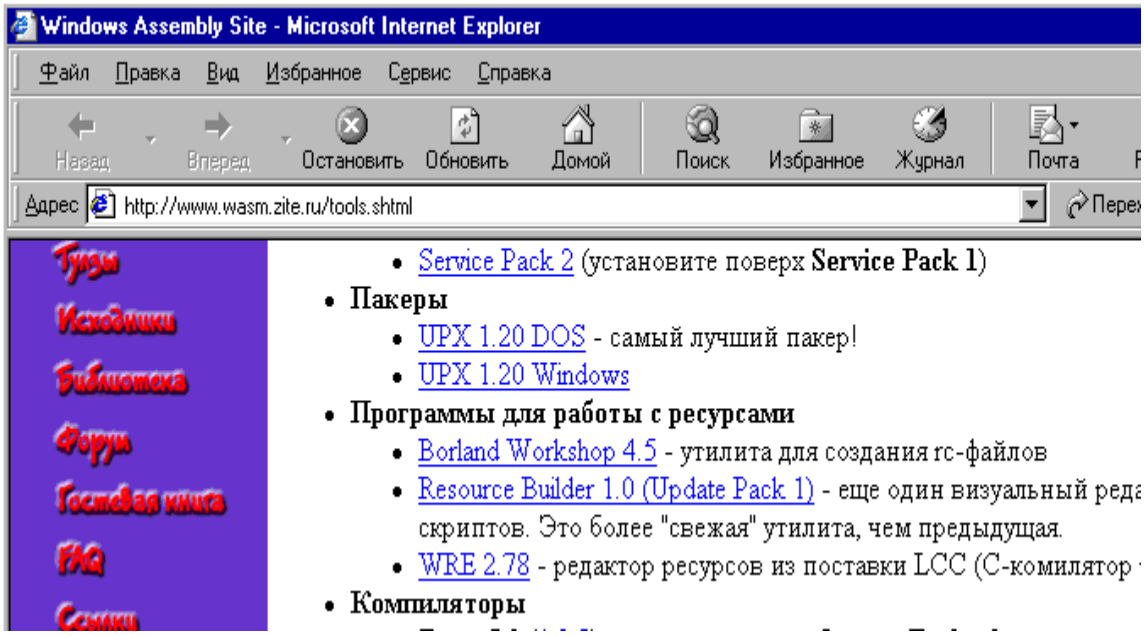
Шаг 356 - EXE файл в ресурсах

http://www.firststeps.ru/mfc/steps/r.php?356

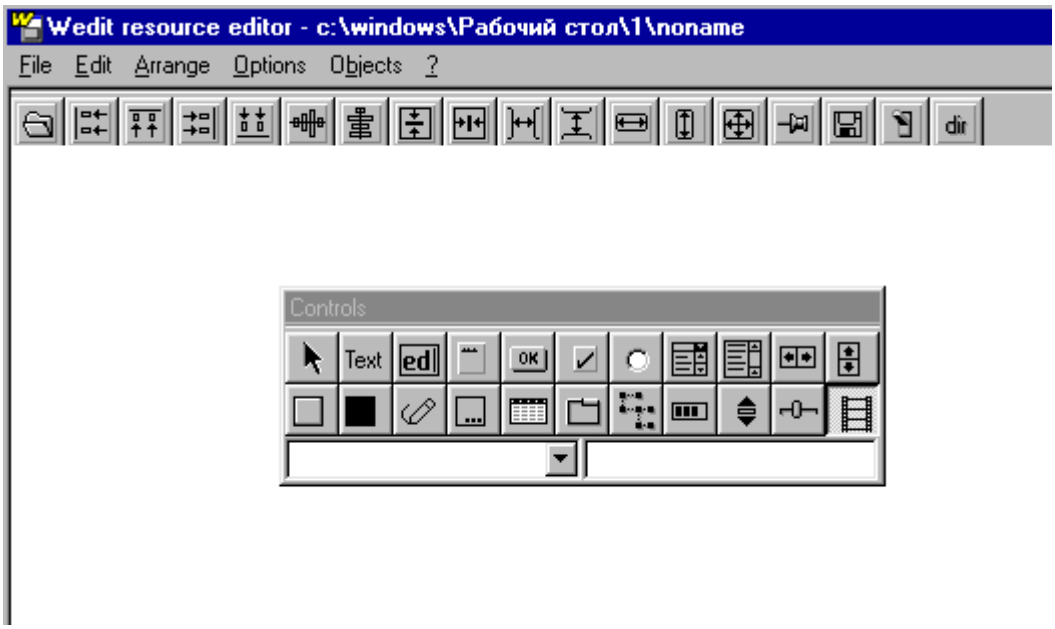
2. Дополнительная информация – в MSDN, на рус. яз.:
http://msdn.microsoft.com/ru-ru/library/74sfy6d5.aspx «System.Resources - пространство имен»
3. Шаг 127 - Программа для создания ресурсов

Итак, создавать ресурсы можно и в **Visual Studio**, но есть и отдельные утилиты. Сходите по адресу:

```
http://www.wasm.zite.ru/tools.shtml
```



Здесь Вы можете взять утилиту **Weditres**. очень маленькую по размеру. Она в архиве, достаточно просто ее распаковать, никакой установки.

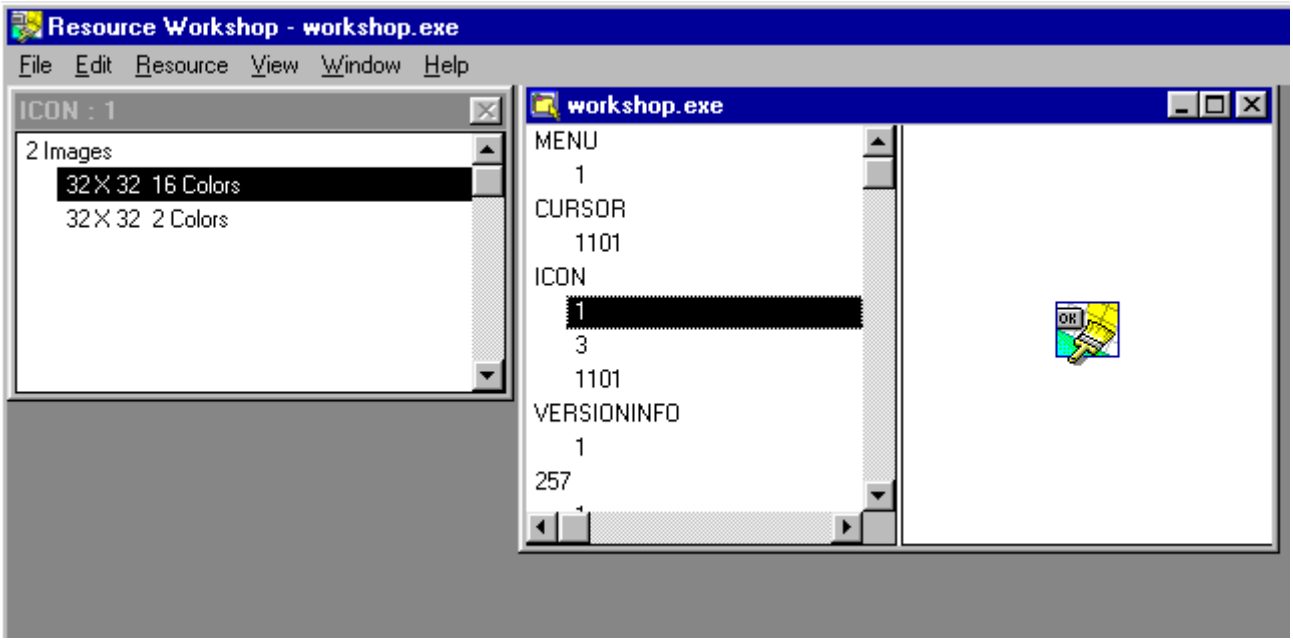


4. Шаг 128 - Еще одна программа для создания и редактирование ресурсов

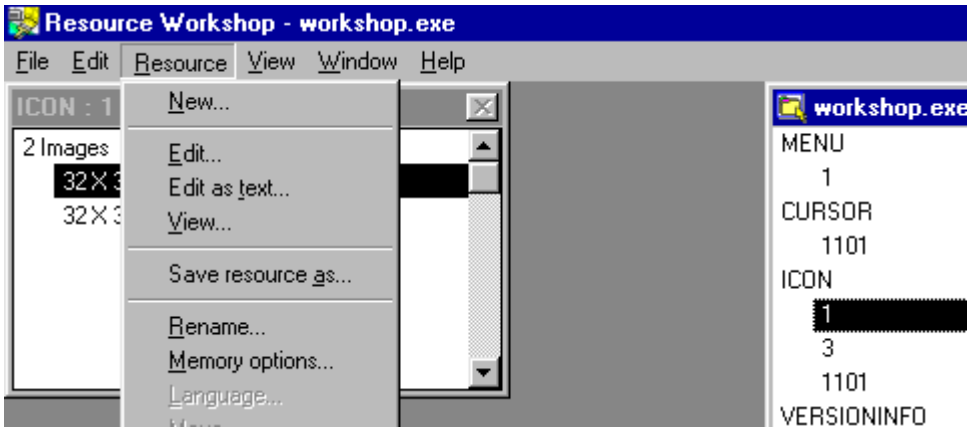
Взять ее Вы можете по адресу:

`http://www.wasm.zite.ru/tools.shtml`

В том же месте, что и в прошлом шаге. Это **Resource WorkShop** из комплекта **Borland C++** в шаге "Шаг 6 - Создание меню". Я уже упоминал о ней. Достаточно распаковать архив и можно использовать. Основная ее прелесть в том, что Вы можете спокойно исследовать любые **EXE** и **DLL** файлы в поисках ресурсов. Вот смотрите.



В окне Выше Вы видите как редактор ресурсов открыл сам себя и позволяет просматривать любые ресурсы. Так как эта утилита входила в среду разработки Вы можете делать с помошью нее очень много.



Пример работы с проектом.

Задание

Напишите в блокноте три программы. Оформите все 3 программы в виде общего dll-файла, из которого можно вызвать на исполнение любую, указав в командной строке её номер. Проверьте работоспособность каждой программы, вызванной из dll. При работе используйте литературу из MSDN.

Пример 1 (создание события).

Напишите консольную программу, в которой программа просит ввести с клавиатуры любое число в диапазоне от 0 до 9.

Если введённое число не равно 0, вырабатывается событие, при обработке которого выводится сообщение «Введено правильное число» и повторяется запрос на ввод нового числа.

При вводе 0 программа должна молча завершиться.

Пример 2 (создание двух событий).

Напишите консольную программу, в которой программа просит ввести с клавиатуры любое число в диапазоне от 0 до 9.

Если введено чётное число, вырабатывается событие, по которому выводится само число и дата.

Если введено нечётное число, вырабатывается событие, по которому выводится само число и строка из какого-нибудь стихотворения.

Пример 3 (одно событие обрабатывается по-разному в разных ресиверах).

Напишите консольную программу, в которой программа просит ввести с клавиатуры любое число в диапазоне от 0 до 9.

Если введено чётное число, ресивер 1 выводит текущую дату и время, а ресивер 2 – пустую строку и затем – строку из какого-нибудь стихотворения. После этого программа завершается.

Если введено нечётное число, запрос числа повторяется.

Ход выполнения:

1. Размещаем каждую из программ в собственном пространстве имен, а именно:

Для 1-ой: namespace Program1Events1

Для 2-ой: namespace Program1Events2

Для 3-ой: namespace Program1Events3

2. Корректируем строки для программ:

для 1-ой

```
  .....
   public class Program1
   {
     public static void Prog1()
 ......
```

для 2-ой

```
  .....
   public class Program2
   {
     public static void Prog2()
 ......
```

для 3-ой

```
  .....
   public class Program3
   {
     public static void Prog3()
 ......
```

4.Сохраняем их по именами **programm1.cs**, **programm2.cs**, **programm3.cs**.

Листинг programm1.cs

```
using System;

namespace Program1Events1
{
    delegate void MyEventHandler();
    class MyEvent
    {
        public event MyEventHandler SomeEvent;
        public void FireSomeEvent()
        {
            if (SomeEvent != null)
                SomeEvent();
        }
    }

    class Reciver
    {
        public static void handler()
        {
            Console.WriteLine("Введено верное число");
        }
    }

    public class Program1
```

```
    {
        public static void Prog1()
        {
            MyEvent evt = new MyEvent();
            evt.SomeEvent += new MyEventHandler(Reciver.handler);
            int cifra = 0;
            do
            {
                Console.WriteLine("Введите число от 0 до 9:");
                cifra = int.Parse(Console.ReadLine());
                if (cifra != 0)
                    evt.FireSomeEvent();
            } while (cifra != 0);
        }
    }
}
```

Листинг programm2.cs

```
using System;

namespace Program1Events2
{
    delegate void MyEventHandler1();
    delegate void MyEventHandler2();
    class Event1
    {
        public event MyEventHandler1 myEvent1;
        public void FiremyEvent1()
        {
            if (myEvent1 != null)
                myEvent1();
        }
    }

    class Event2
    {
        public event MyEventHandler2 myEvent2;
        public void FiremyEvent2()
        {
            if (myEvent2 != null)
                myEvent2();
        }
    }

    class Reciver1
    {
        public static void handler1()
        {
            DateTime d = DateTime.Now;
            Console.WriteLine(d);
        }
    }

    class Reciver2
    {
        public static void handler2()
        {
            //Console.WriteLine("Введено число  " + cifra);
            Console.WriteLine("Белеет парус одинокий");
        }
    }
    public class Program2
    {
        public static void Prog2()
```

```
        {
            Event1 evt1 = new Event1();
            evt1.myEvent1 += new MyEventHandler1(Reciver1.handler1);
            Event2 evt2 = new Event2();
            evt2.myEvent2 += new MyEventHandler2(Reciver2.handler2);

            int cifra = 0;
            Console.WriteLine("Введите число от 0 до 9:");
            cifra = int.Parse(Console.ReadLine());

            if ((cifra == 0) | (cifra == 2) | (cifra == 4) | (cifra == 6) | (cifra == 8))
            {
                Console.WriteLine("Введено число  " + cifra);
                evt1.FiremyEvent1();
            }
            else
            {
                Console.WriteLine("Введено число  " + cifra);
                evt2.FiremyEvent2();
            }

        }
    }
}
```

Листинг programm3.cs

```
using System;

namespace Program1Events3
{
    delegate void MyEventHandler1();
    class Event1
    {
        public event MyEventHandler1 myEvent1;
        public void FiremyEvent1()
        {
            if (myEvent1 != null)
                myEvent1();
        }
    }


    class Reciver1
    {
        public static void handler1()
        {
            Console.WriteLine("Работает обработчик 1");
            DateTime d = DateTime.Now;
            Console.WriteLine(d);
        }
    }

    class Reciver2
    {
        public static void handler2()
        {
            Console.WriteLine("Работает обработчик 2");
            Console.WriteLine("");
            Console.WriteLine("Белеет парус одинокий");
        }
    }
    public class Program3
    {
        public static void Prog3()
```

```
        {
            Event1 evt1 = new Event1();
            evt1.myEvent1 += new MyEventHandler1(Reciver1.handler1);
            evt1.myEvent1 += new MyEventHandler1(Reciver2.handler2);

            int cifra = 0;
        A:
            Console.WriteLine("Введите число от 0 до 9:");
            cifra = int.Parse(Console.ReadLine());

            if ((cifra == 0) | (cifra == 2) | (cifra == 4) | (cifra == 6) | (cifra == 8))
            {
                Console.WriteLine("Введено число  " + cifra);
                evt1.FiremyEvent1();
            }
            else
                goto A;
        }
    }
}
```

5.Собираем библиотеку **lib.dll** из указанных исходников, используя командную строку:

csc /out:lib.dll /t:library programm1.cs programm2.cs programm3.cs

6. Пишем главную программу.

Листинг glavnaja.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Program1Events1;
using Program1Events2;
using Program1Events3;

namespace Glavn
{
    class Program
    {
        static void Main(string[] args)
        {
            int nomprog;
            do
            {
                Console.WriteLine("Введите номер программы от 1 до 3:");
                nomprog = int.Parse(Console.ReadLine());
                switch (nomprog)
                {
                    case 1:
                        Console.WriteLine("Вызвана программа №1");
                        Program1.Prog1();
                        break;

                    case 2:
                        Console.WriteLine("Вызвана программа №2");
                        Program2.Prog2();
                        break;

                    case 3:
                        Console.WriteLine("Вызвана программа №3");
                        Program3.Prog3();
                        break;
```

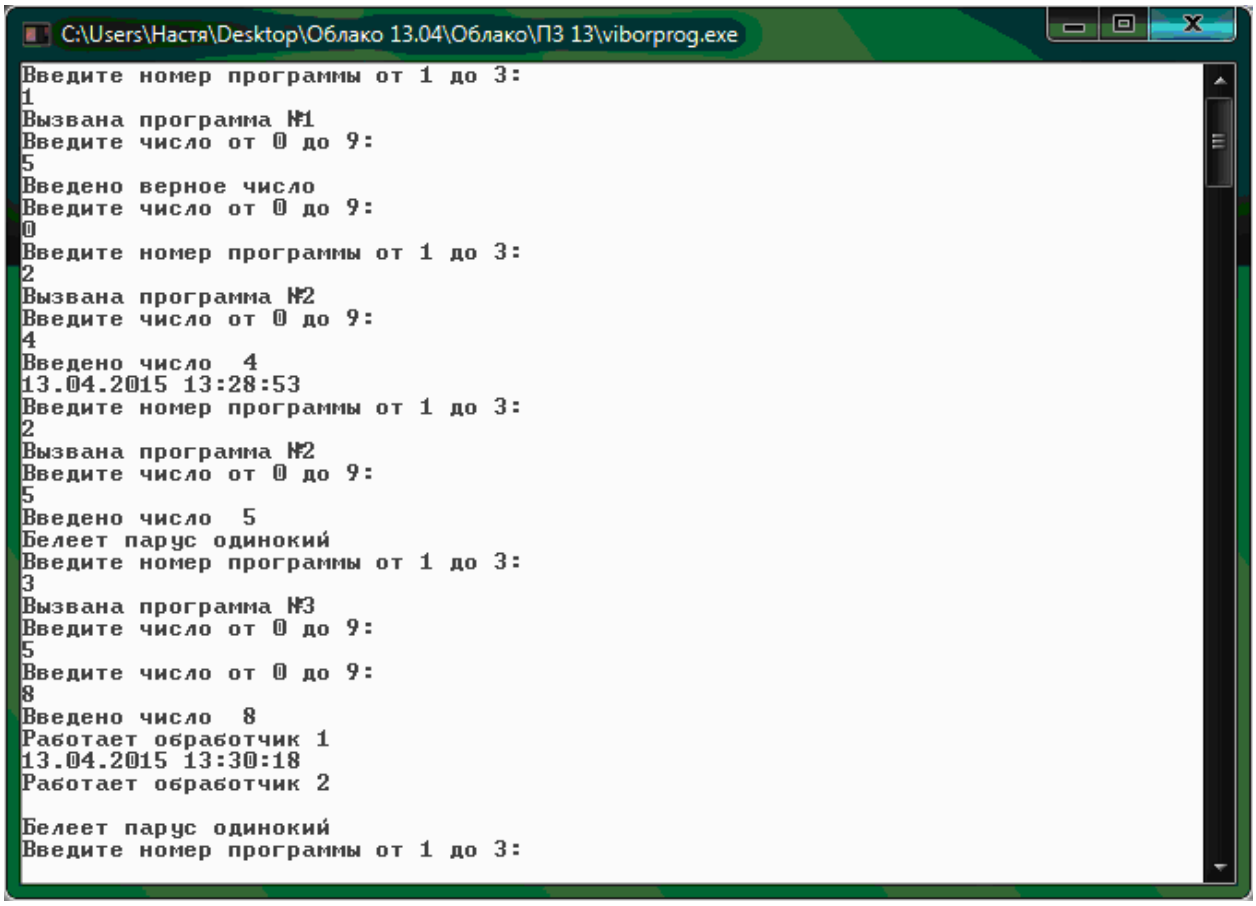
```
                    default:
                        Console.WriteLine("Указан несуществующий номер программы");
                        Console.ReadLine();
                        break;
                }
            } while (nomprog >= 1 && nomprog <= 3);
        }
    }
}
```

7. Собираем загрузочный модуль программы используя командную строку:

csc /out:viborprog.exe /reference:lib.dll glavnaja.cs

8.Запускаем и проверяем.

Результат выполнения программы



Литература.

1. Объектно-ориентированное программирование на алгоритмическом языке C#, Кириченко А. А. // М. : Государственный университет - Высшая школа экономики, 2015.
2. Типы данных / Данные в языках программирования., Леман Д., Смит М. // М.: Мир, 1982.
3. Теория синтаксического анализа, перевода и компиляции, Т.1 "Синтаксический анализ",А. Ахо, Дж. Ульман // М.: Мир, 1978

4. Основы конструирования компиляторов. Лексический анализ на C# [Электронный ресурс]// URL: https://habrahabr.ru/post/132422/ (Дата обращения: 10.04.2015, режим доступа: свободный).

5. Уоссермен Ф. Нейрокомпьютерная техника : теория и практика. М. : Мир, 1992. – 240 с.
6. А.П.Пятибратов, Л.П Гудыно, А.А.Кириченко «Вычислительные системы, сети и телекоммуникации», ИНФРА-М, 2008. – 736с.:ил.
7. Нейронные сети Statistica Neural Networks. Методология и технологии современного анализа данных. Под ред. Боровикова В.П., М., Горячая линия – Телеком, 2008.
8. Н. М. Абдикеев «Проектирование интеллектуальных систем в экономике», М., 2003.
9. Методы добычи данных. Internet-ресурс – http://www.statsoft.ru
10. Теория синтаксического анализа, перевода и компиляции, Т.1 "Синтаксический анализ",А. Ахо, Дж. Ульман // М.: Мир, 1978
11. Основы конструирования компиляторов. Лексический анализ на C#[Электронный ресурс]// URL: https://habrahabr.ru/post/132422/ (Дата обращения: 10.04.2015, режим доступа: свободный).
12. Объектно-ориентированное программирование на алгоритмическом языке C#, Кириченко А. А. // М. : Государственный университет - Высшая школа экономики, 2015.
13. Типы данных / Данные в языках программирования., Леман Д., Смит М. // М.: Мир, 1982.
14. Теория синтаксического анализа, перевода и компиляции, Т.1 "Синтаксический анализ",А. Ахо, Дж. Ульман // М.: Мир, 1978
15. Основы конструирования компиляторов. Лексический анализ на C#[Электронный ресурс]// URL: https://habrahabr.ru/post/132422/ (Дата обращения: 10.04.2015, режим доступа: свободный).

Монография (сетевое электронное издание). 152 страницы, формат PDF.

ISBN 978-5-9904911-6-8

Кириченко А.А.
профессор департамента программной инженерии факультета компьютерных наук Федерального государственного автономного образовательного учреждения высшего профессионального образования “Национальный исследовательский университет "Высшая школа экономики" ”.

«Программное обеспечение нейросетевых исследований»

Редактор Шенникова З.И.

Издательство "Высшая школа экономики".
2016г.