МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

НР
НАУКА В РЕГИОНЫ

Методические рекомендации к курсу дополнительного общего образования «ПРОЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ» для первого года обучения

Фонд развития
Физтех-лицея и Физтех-школ

Иннопрактика

МФТИ
Долгопрудный, 2021

Автор:
Дединский Илья Рудольфович
Старший преподаватель
кафедры информатики МФТИ

С О Д Е Р Ж А Н И Е

Профессионально-ориентированный подход
к довузовскому преподаванию программирования 3

Итоги педагогического эксперимента по внедрению проектного подхода
в профессионально-ориентированном обучении программированию 19

Вводная документация к библиотеке TX Library 28

Профессионально-ориентированный подход к довузовскому преподаванию программирования

IT-индустрия в настоящее время является одной из глобальных системообразующих отраслей, её развитие во многом определяет технологическое развитие других областей. Характерной чертой IT-индустрии является сильная зависимость от уровня компетенций специалистов (человеческого фактора), которая, в свою очередь, зависит от уровня профессионального образования. Высокая скорость развития информационных технологий, прежде всего программирования и разработки ПО, делает крайне проблематичным серьёзное изучение профессии начиная лишь с ВУЗа – зачастую от тех, кто начинал ещё со школы, наивный первокурсник уже отстал навсегда. Поэтому для успешного состояния IT-отрасли критично необходимо развивать образование довузовское, как это происходит в областях, близких к программированию – математике и физике, и где старт углублённого обучения по факту происходит уже в 6-8 классе школы.

Российское довузовское образование в настоящее время содержит некоторую возможность углубления в области информационных технологий. Если оно связано с углублённой математикой и физикой (а не предполагает лишь офисные технологии), то оно может приносить плоды и для развития IT-индустрии. Однако нынешние тенденции такого образования в России на данный момент расходятся как с современными требованиями к компетенциям IT-специалистов, так и с образовательными подходами за рубежом, что порождает странную на первый взгляд картину, когда в соревнованиях по спортивному программированию участники

из России часто занимают весьма высокие позиции, а в то же время профессиональное сообщество российских программистов испытывает сильный недостаток высококвалифицированных кадров. В результате мы повсеместно пользуемся импортированным в Россию программным обеспечением и технологиями, что не добавляет российскому технологическому пространству ни развития, ни стабильности. Российские IT-специалисты, а вслед за ними и компании, в массе своей конкурентно проигрывают зарубежным по уровню компетенций, но совсем не потому, что имеют худшие способности. Просто их *«так учили»*.

Углублённый подход к преподаванию информатики в большинстве случаев применяется в учебных заведениях выраженной физико-математической направленности и предполагает курс программирования, что безусловно правильно. Проблема в том, что в большинстве случаев способом реализации курса является решение большого количества изолированных алгоритмических задач (так называемый «олимпиадный подход», часто называемый «алгоритмическим»).

Однако, если ограничиваться только таким подходом и игнорировать современные тенденции развития IT-отрасли, зачастую получается, что даже успешный олимпиадник испытывает серьёзные проблемы с успешностью при попытках выйти за пределы олимпиадной стилистики, например, при вступлении в профессию. Так, по данным опросов Центра педагогического мастерства (Москва) две трети победителей и призёров Всероссийских олимпиад школьников 2015 года затруднились ответить на вопрос «Кем вы видите себя по профессии после окончания вуза?». Среди победителей Всероссийской олимпиады по информатике только 25-35% указали возможную профессию «программист».

Существующее доминирование чисто спортивных тенденций практически не учитывает тот факт, что участие в разработке ПО, как для научных целей, так и в качестве инженерной профессии, – процесс проектно-ориентированный, а это требует многих качеств, которые в олимпиадном (алгоритмическом) подходе не нужны и, как следствие, не развиваются. Безусловно, владение алгоритмическими основами обязательно для будущего профессионала, однако его опыт никак не должен этим исчерпываться (***см. рис. 1***).

Олимпиадный подход, вследствие формально-соревновательного, зачастую спортивного характера, доминируя сейчас в российском образовательном пространстве, преследует в основном внутренние цели, необходимые лишь для реализации предельно краткого жизненного цикла решения (написал – сдал – забыл), и которые лишь частично коррелируют с профессиональными компетенциями, необходимыми для длительного жизненного цикла разработки ПО. Иногда эти цели даже противоречат таким компетенциям (например, умению конструировать грамотный инженерно-технический компромисс), что даёт описанный выше диссонанс между спортивными успехами и плачевным состоянием дел в российский индустрии. Полностью олимпиадный подход соответствует только одной профессии – олимпиадного тренера (или организатора олимпиад), что делает эту систему замкнутой.

В то же время профессионально-ориентированный (проектный) подхода совсем не противоречит олимпиадному движению, так проектный подход включает и концепции автоматического тестирования, которые приняты за основу олимпиадных технологий в информатике. Фактически, алгоритмический подход – основа олимпиад – это одна из частей проектного подхода. Однако на данный момент подавляющее доминирование именно спор-

Рис. 1. Модели опыта и особенности динамики его набора в проектном и олимпиадном подходах. «Граблями» обозначены трудности, в ходе которых набирается необходимый опыт. См. [6].

тивной составляющей мешает развитию профессиональной ориентированности, так что дело не в антагонизме, а в восстановлении паритета.

В результате характерный для каждой профессии диссонанс между *«тем, чему учили»* и *«тем, что необходимо в профессии»*, описывается весьма непустым множеством образовательных разрывов, которые в настоящее время учащийся и студент должен преодолевать сам, и которые составляют его личный опыт. Такая ситуация существует и в школе, и в ВУЗе уже очень долго. В то же время, большинство разрывов типичны и легко обнаруживаются в ходе внимательного анализа (***см. рис. 2***).

Цель настоящего подхода – проанализировать образовательные разрывы и построить курс таким образом, чтобы минимизировать эти разрывы и максимизировать набор конструктивного положительного опыта, не ограничивающимся лишь конкретными приёмами, шаблонами и средствами. Это позволяет учащимся в дальнейшем ориентироваться в динамичном мире информационных технологий, которые часто успевают развиться и умереть до того, как по ним выйдет первый учебник. В таких условиях главная учебная задача, и не только в сфере ИТ, – научить будущего студента действовать грамотно и самостоятельно. Под грамотностью здесь понимается умение классифицировать проблемы, знать типовые решения, выбирать из них спектр адекватных решений, комбинировать их, придумывать новые решения, контролировать качество, мыслить не рецептами, а как минимум технологиями.

Для этого автором вводится понятие когнитивно-технологической единицы (КТЕ), как единицы действительного усвоения знаний, определенной следующим образом [1]:

Рис. 2. Образовательные разрывы между источниками знаний. См. [6].

- Зачем это надо,
- Что это такое,
- На чём основано и с чем связано,
- Как это применять,
- Где это можно и где нельзя использовать,
- Чем придется пожертвовать,
- Что будет, если этого не делать,
- Какие в этом «подводные камни» (чего опасаться при применении).

Разработанный курс рассчитан на математически сильных и высоко работоспособных учащихся 7(8) – 10(11) классов физико-математического профиля, нагрузку 4-6 профильных учебных часа в неделю, развитую систему факультативов и консультаций. В преподавании используются принципы, отличающиеся от алгоритмического (олимпиадного) подхода (***см. табл. 1***):

Важнейшей задачей курса является формирование системы профессиональных ценностей (предпочтений) ученика (***см. рис. 3***). В конёчном счёте, это формирование и есть основная инвариантная методологическая задача курса, так как всё остальное – лишь технология и будет неотвратимо изменяться с течением времени.

Табл. 1. Сравнение особенностей проектного и олимпиадного (алгоритмического) подходов.

Проектный подход	Олимпиадный (алгоритмический) подход
Главными методологическим принципом является системный подход.	Главными принципом является соревновательный подход с судейством по формальным правилам.
Во главу угла ставится задача, понимаемая как часть проекта, и, главное, путь от задачи к решению, а не кодирование алгоритма. Полученные решения повторно используются в дальнейшем, как и в реальной индустрии, что заставляет их реализовывать более системно и внимательно, увязывать с другими решениями. Работа основана на реальном (длительном) жизненном цикле разработки ПО.	Рассматриваются изолированные задачи, общие лишь по алгоритмической тематике. Использование реализаций полученных решений запрещено в последующих задачах. Сборки решений в цельный проект не предусматривается. Жизненный цикл решения предельно краток (написал – сдал).
Проектный подход включает в себя тестирование реализованных алгоритмов (т.е. того, на чём основан олимпиадный подход), что соответствует положению дел в индустрии.	Олимпиадный подход не включает проектный и зачастую пытается рассматривать его как ненужного конкурента.
Технологичность, надежность и масштабируемость решения ставятся выше *«программистских трюков»*, иногда позволяющих в отдельных случаях добиться несколько лучших результатов. Активно рассматриваются альтернативные решения, полезные при масштабировании проекта.	Если для задачи допустимо неуниверсальное, немасштабируемое, «неинженерное» (и иногда по сути неверное), но оно формально допустимо, то оно всё равно годится для соответствия конкретным условиям соревнования. Рассмотрение альтернативных решений в этом виде образовательного процесса вторично.

Проектный подход	Олимпиадный (алгоритмический) подход
Понимание и корректное использование учащимся тех средств, с помощью которых он решил задачу, ставится выше уровня самих средств решения. Выполняется рецензирование исходного текста преподавателем, сдача работы производится в форме собеседования. Подход к оцениванию – комплексный.	Исходный текст программы и технологичность решения не оценивается (его смотрят только при подозрении на прямой плагиат), тестирование ведётся без участия преподавателя. Единственный критерий оценки – количество автоматических тестов, пройденных программой. Собеседование по итогам решения не влияет на оценку.
Самостоятельность решения является ключевым условием, которое необходимо доказать при сдаче работы.	Системы тестирования решения способны детектировать прямой плагиат, но заимствование с частичной переработкой детектировать не способны.
Для записи алгоритма на языке программирования выбирается подмножество наиболее универсальных средств языка, чтобы не акцентировать внимания на кодировании и для более лёгкого перехода на другие языки программирования.	Активно используются особенности конкретных языков программирования, позволяющих легче решить определённые задачи, что «привязывает» учащихся к конкретным языкам и технологиям, которые завтра изменятся.
Задачи ставятся в нескольких вариантах различной сложности (от базового до творческого), при сдаче работы засчитывается решение на любом уровне. Уровень сложности фиксируется и используется как дополнительная информация к оценке, для выяснения и повышения уровня профессионализма ученика.	Понятие творческого подхода не существует в силу предельной формализации критериев оценивания, однако существует деление задач по их сложности.

Проектный подход	Олимпиадный (алгоритмический) подход
В обучении активно применяются современные парные и групповые техники, использующиеся в индустрии (обмен кодом и документацией, перекрестное рецензирование и тестирование, групповая разработка стандартов взаимодействия компонентов проекта). Эти же техники используются при подготовке к ЕГЭ по информатике.	Групповые техники формально существуют, но участники контеста делятся либо по задачам (каждый решает свою), либо по «специализациям» – один знает больше алгоритмов, другой быстрее отлаживает программы и т.п. Такие техники не способствуют объединения учебных навыков у участников. Не развиваются технологии настоящей групповой работы.

Подход, ориентированный на проектную работу, сильно увлекает многих учеников и даёт не только высокие проектные результаты (призовые места на Всероссийских и международных конкурсах – финале Intel Scientific and Engineering Fair (ISEF), Балтийском научно-инженерном конкурсе, конкурсах «Ученые будущего», «Интел-Юниор», «Интел-Династия-Авангард», «Старт в науку», «Шаг в науку»), но и высокие олимпиадные (победителей и призёров Всероссийского и международного уровня). Однако, так как он не совпадает с распространённым сейчас олимпиадным подходом, то он не всегда одобряется приверженцами олимпиад, которые зачастую хотят получить ученика «целиком и полностью», проявляя ярко выраженный монополизм. Они часто воспринимают проектную работу высокого уровня как конкуренцию, оттягивающую от них сильных детей, и пытаются создать для неё неблагоприятную среду, в том числе необъективными и не всегда корректными методами. Тем не менее, по-настоящему сильные олимпиадные школы России видят в проектно-профессиональном подходе большую перспективу (см., напр., письма руководителей и сотрудников широко известных

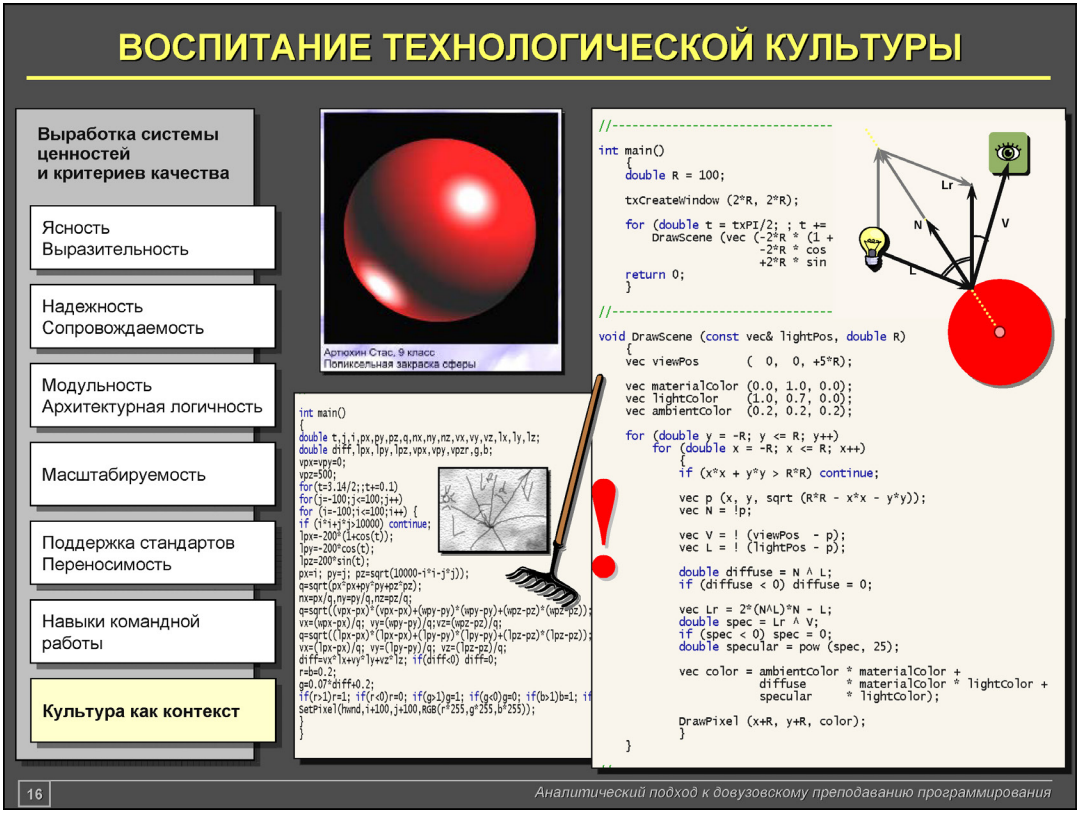


Рис. 3. Выработка системы профессиональных ценностей учащегося.
В центре снизу показан фрагмент программы типичного студента, занимавшегося олимпиадным подходом.
Справа представлен фрагмент программы девятиклассника, второй год обучавшегося по проектной методике.
Алгоритмы программ идентичны (попиксельная закраска трехмерной сферы). См. [6].

в программировании и прикладной математике университетов СПбГУ ИТМО [2, 3], МФТИ (ГУ) [4]). Зарубежные учителя информатики также признают этот подход крайне плодотворным [5].

Результатом прохождения курса становится не только понимание основных принципов программирования и владение основными алгоритмическими конструкциями, но и серьёзные концептуальные и технологические навыки, позволяющие самостоятельно разрабатывать проекты достаточно большого для школьников объема (порядка курсовой работы 2-3 курса ВУЗа, а иногда и бакалаврского диплома), успешно работать в групповых проектах, требующих активного взаимодействия участников, а некоторым – участвовать и регулярно побеждать в различных конкурсах и олимпиадах Всероссийского и международного уровней, участвовать в научных конференциях РАН наравне со взрослыми (***рис. 4***).

Пример: в 2015 году на пилотной смене в образовательном центре «Сириус» (г. Сочи) проектные работы учеников были представлены В. В. Путину как главные научные достижения смены и были высоко оценены экспертами из правительства России, МФТИ и ИППИ РАН [7] (***рис. 5***).

К сожалению, на данный момент чётко работающего методического «канала», который закономерно привёл бы увлеченного школьника к профессии программиста, в России нет. Существующее дополнительное образование эту задачу не решает в силу его нерегулярности и частой методической несогласованности: мотивируемые личными, часто лишь финансовыми, амбициями, разнообразные ведущие кружков, сборов и курсов пытаются тянуть одеяло на себя. Рынок дополнительного образования по информатике и программированию чрезвы-

Рис. 4. Статистика результатов руководителя педагогической образовательной площадки И. Р. Дединского. за время проведения педагогического эксперимента (2005–2009 г.г.). См. [6].

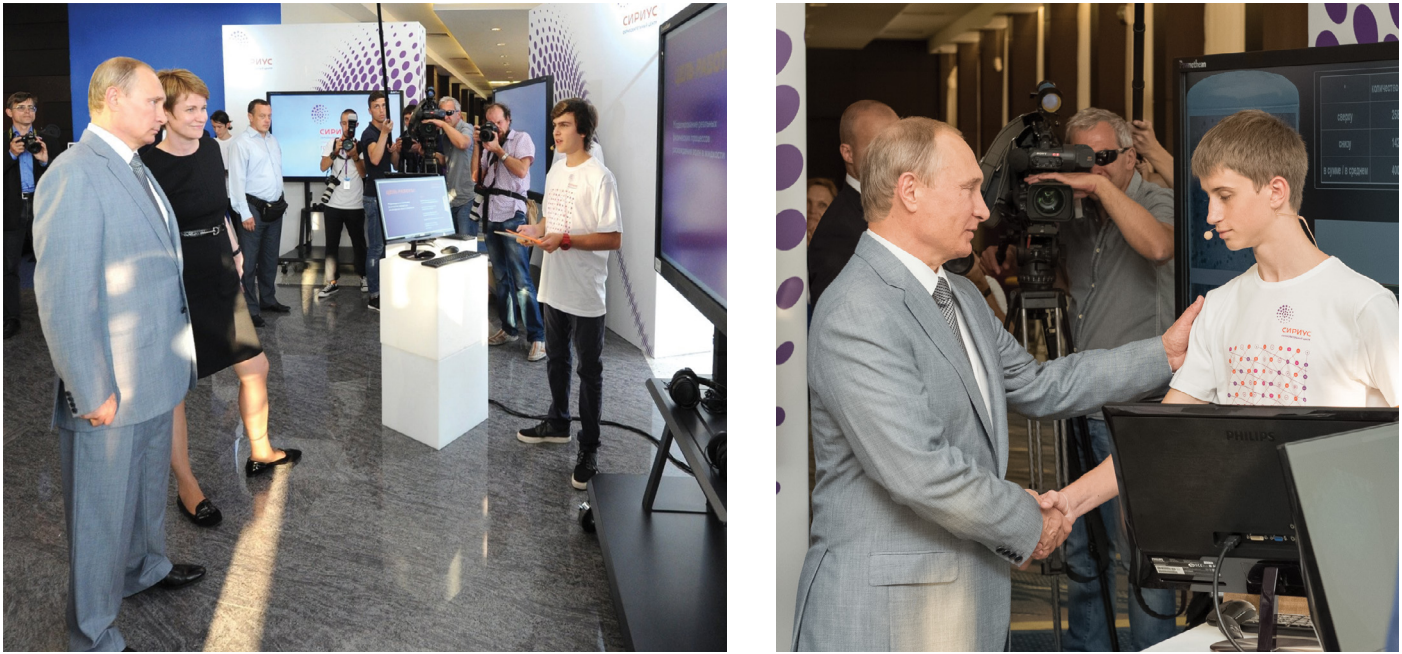


Рис. 5. Проектные работы школьников представлены В. В. Путину в образовательном центре «Сириус» (г. Сочи). См. [7].

чайно перенасыщен ворохом, скажем так, сомнительных предложений на хайповые темы: вот уже чуть ли не младших школьников учат программировать с использованием дополненной реальности, анализа больших данных и машинного обучения. Главное, конечно, не спрашивать детей, как работает та или иная функция из огромного количества сторонних

библиотек, которые подсовывают детям педагоги с низкой социальной ответственностью, прекрасно понимая, что заставляют детей работать лишь на самом примитивном уровне — уровне воспроизведения. Конечно, зачем разбираться в деталях, когда «и так работает»?

Всё это ведет к халтуре как части характера ребенка, подходу «и так сойдёт», гигантскому дистанцированию его от профессионального подхода, а довольный «педагог» при этом уже подсчитывает материальные или социальные барыши. Такая педагогическая и методическая халтура, часто сознательная, порождает не обучение, а лишь иллюзию обучения, причём опасную: ребёнок далее ошибочно считает, что во многих областях он уже профи, и это существенно снижает мотивацию для более глубокого обучения впоследствии и обрекает его на трудное переучивание, редко успешное. Ребёнок, лишь ставший взрослым, поймёт, что его цинично обманули. В этом информационном мусоре трудно разобраться и ребёнку, и родителям. Программисты в России растут «как трава», «как-то сами», и поэтому их так мало и из них ещё меньше хороших.

Огромный спрос на высокопрофессиональных программистов в России, подтверждаемый многими работодателями, подтверждает печальные последствия попыток циничных олимпиадных функционеров и жадных до прибыли эксплуататоров хайповых трендов установить удобные лишь им правила игры, игнорируя то, ради чего программирование было создано и существует. Представленный подход способен решить вопрос наличия такого «канала», с помощью которого можно получить большое количество высококлассных специалистов для развития российской индустрии информационных технологий.

Литература

1. *И. Р. Дединский. Как хотеть учиться. // Компьютерра. – 2005, № 24 (тема номера: «Тьма просвещения»). https://old.computerra.ru/2005/596/207625/.*
2. *Проф. В. Н. Васильев, проф. В. Г. Парфенов, проф. А. А. Шалыто, А. С. Станкевич, Г. А. Корнеев (СПбГУ ИТМО). Письмо об оценке работы экспериментальной образовательной площадки. // 2009, http://ded32.net.ru/news/2009-04-04-32.*
3. *Проф. В. Г. Парфенов, проф. А. А. Шалыто (СПбГУ ИТМО). Письмо в защиту инновационной работы. // 2010, http://ded32.net.ru/news/2010-09-10-55.*
4. *Проф. А. А. Шананин, проф. И. Б. Петров (МФТИ). Оценка работы экспериментальной площадки. // 2009, http://ded32.net.ru/news/2009-09-01-39.*
5. *И. Р. Дединский. Курсы по методике преподавания программирования для учителей Южной Кореи. // 2009, http://ded32.net.ru/news/2009-08-20-37.*
6. *И. Р. Дединский. Аналитический подход к довузовскому преподаванию программирования – принципы преподавания и итоги эксперимента. Тезисы доклада и презентация. // Труды Всероссийского съезда учителей информатики в МГУ. – 2011, М., Изд-во МГУ. http://ded32.net.ru/news/2011-04-03-58.*
7. *И. Р. Дединский. Проектные работы по программированию представлены Президенту России В. В. Путину. // 2015, http://ded32.net.ru/news/2015-09-02-77.*

Итоги педагогического эксперимента по внедрению проектного подхода в профессионально-ориентированном обучении программированию

Углублённый подход к преподаванию информатики в большинстве случаев применяется в учебных заведениях или группах физико-математической направленности и предполагает курс программирования, что связано с дальнейшим обучением по этому профилю в ВУЗе. В большинстве случаев способом реализации курса является решение большого количества изолированных алгоритмических задач (так называемый олимпиадный подход).

Однако, если ограничиваться только этим и игнорировать современные тенденции развития процесса разработки программного обеспечения, может получиться, что даже успешный олимпиадник будет испытывать серьёзные проблемы с успешностью при попытках выйти за пределы олимпиадной стилистики. Это связано с тем, что участие в разработке ПО, как для научных целей, так и в качестве инженерной профессии – процесс проектно-ориентированный, а это требует многих качеств, которые в олимпиадном подходе не нужны и, как следствие, не развиваются.

В результате характерный для каждой профессии диссонанс между «тем, чему учили», «тем, что надо в работе», описывается непустым множеством образовательных разрывов, которые в настоящее время учащийся и студент должен преодолевать сам, и которые составляет его личный опыт. Такая ситуация существует и в школе, и в ВУЗе. В то же время, большинство разрывов типичны и легко обнаруживаются в ходе внимательного анализа.

Цель данной работы – проанализировать образовательные разрывы и построить курс таким образом, чтобы минимизировать эти разрывы и максимизировать набор конструктивного положительного опыта, не ограничивающимся лишь конкретными приёмами, шаблонами и средствами. Это позволяет учащимся в дальнейшем ориентироваться в меняющемся мире информационных технологий, которые часто успевают развиться и умереть до того, как по ним выйдет первый учебник. В таких условиях главная учебная задача, и не только в сфере ИТ, – научить студента действовать грамотно и самостоятельно. Под грамотностью здесь понимается умение классифицировать проблемы, знать типовые решения, выбирать из них спектр адекватных решений, комбинировать их, придумывать новые решения, контролировать качество, мыслить не рецептами, а как минимум технологиями [1].

Для этого автором вводится понятие когнитивно-технологической единицы (КТЕ), как единицы действительного усвоения знаний, определенной следующим образом [2]:

- *Зачем это надо,*
- *Что это такое,*
- *На чём основано и с чем связано,*
- *Как это применять,*
- *Где это можно и где нельзя использовать,*
- *Чем придётся пожертвовать,*
- *Что будет, если этого не делать,*
- *Какие в этом «подводные камни» (чего опасаться).*

Разработанный курс рассчитан на учащихся 7(8) – 10(11) классов, нагрузку минимум 4 учебных часа в неделю и систему факультативов. Он учитывает разнородную предварительную подготовку учащихся, и тот факт, что часть из них не изучали информатику и программирование вовсе. По этой причине в начале курса преподавание ведётся «с нуля», в предположении, что учащийся не обладает какими-либо специальными знаниями в области программирования. По этой причине используются следующие принципы:

- *Во главу угла ставится задача, понимаемая как часть проекта, и, главное, путь от задачи к решению, а не кодирование алгоритма.*
- *Для записи алгоритма на языке программирования выбирается минимальное подмножество средств языка, чтобы не акцентировать внимания на кодировании и для более лёгкого перехода на другие языки программирования.*
- *Самостоятельность решения является ключевым условием, которое необходимо доказать при сдаче работы.*
- *Понимание учащимся тех средств, с помощью которых он решил задачу, ставится выше уровня самих средств решения.*
- *Аккуратность и надёжность решения ставятся выше «программистских трюков», иногда позволяющих в отдельных случаях добиться несколько лучших результатов.*
- *Задачи ставятся в нескольких вариантах различной сложности (от базового до творческого), при сдаче работы засчитывается решение на любом уровне (но удовлетворяющее*

п. 2-4). Уровень сложности фиксируется и используется как дополнительная информация к оценке, для выяснения и повышения уровня профессионализма ученика.

- *Главным методологическим принципом является системный подход.*
- *В обучении активно применяются парные и групповые техники (обмен кодом и документацией, перекрёстные peer review и тестирование, групповая разработка стандартов взаимодействия участников проекта). Эти же техники используются при подготовке к ЕГЭ по информатике.*

Важнейшей задачей курса является формирование системы профессиональных ценностей (предпочтений) ученика. В конечном счёте, это формирование и есть основная инвариантная методологическая задача курса, так как всё остальное – технология и будет неотвратимо изменяться с течением времени.

Принятый подход, ориентированный на проектную работу, сильно увлекает многих учеников и даёт не только высокие проектные результаты (призовые места на Всероссийских и международных конкурсах), но и высокие олимпиадные (победителей и призеров Всероссийского и регионального уровня). Однако надо отметить, что он не совпадает с традиционным подходом (ставящим во главу угла олимпиадное программирование) и не всегда одобряется приверженцами чисто олимпиадного подхода, которые зачастую хотят получить ученика «целиком и полностью» и воспринимают проектную работу такого уровня как конкуренцию. Тем не менее, действительно сильные олимпиадные школы России видят в нём большую перспективу (см., напр., письмо СПбГУ ИТМО).

Результатом прохождения курса становится не только понимание основных принципов программирования и владение основными алгоритмическими конструкциями, но и серьёзные концептуальные и технологические навыки, позволяющие самостоятельно разрабатывать проекты достаточно большого для школьников объёма (порядка курсовой работы 2-3 курса ВУЗа), успешно работать в групповых проектах, требующих активного взаимодействия участников, а некоторым – участвовать и побеждать в различных конкурсах Всероссийского и международного уровней, участвовать в научных конференциях РАН.

Методическое обеспечение, разработанное для поддержки курса

Для первых двух лет обучения (7-8, 8-9 класс), где мотивация наиболее критична, автором разработана компактная библиотека двумерной графики для **Win32** на C++, намеренно выдержанная в стиле сугубого минимализма (**TX Library**). Это небольшая «песочница» для начинающих, реализованная с целью помочь им в изучении простейших принципов программирования. Она также является методическим учебным пособием для обучения основам программирования на C++. Библиотека позволяет писать прямолинейный графический код, не заботясь о событийной модели приложений в **Win32**. Имеется система помощи на русском языке, не требуется компоновки с внешними библиотеками.

Философия **TX Library** — облегчить первые шаги в программировании и подтолкнуть к творчеству и самостоятельности. Исходный текст библиотеки может использоваться для иллюстрации элементарных приемов работы с окнами **Windows**, механизмом сообщений **Win32,** графикой, работой с меню, растровыми образами, простейшей многопоточностью.

Предупреждение, или TXLib — это всего лишь инструмент

Библиотека **TXLib** — это всего лишь инструмент для того, чтобы облегчить первые шаги в программировании. Однако этот инструмент, как и любой другой, может быть применён неправильно. (Тем не менее, в основу **TXLib** заложены некоторые принципы, помогающие конструктивному неиллюзорному обучению.)

Сама по себе любая библиотека или любой язык программирования не научит начинающего писать программы грамотно. Научит этому разработка своих, достаточно больших проектов, в сочетании с тесным общением профессионалов, желающих помочь начинающим. Такие профессионалы должны обладать и опытом разработки больших программ, и педагогическими навыками, чтобы передать свой опыт начинающим. К сожалению, не всегда это совпадает.

Профессионалы-программисты зачастую не хотят лезть в обучение, где хватает своих проблем. Чего только стоят попытки свести воедино точки зрения ребенка, его одноклассников, его родителей и родителей одноклассников, учителя информатики и его коллег, администрации школы, вышестоящего руководства. Это труд, требующий терпения Сизифа, гения стратега и мастерства профессиональных переговорщиков. Автору практически неизвестны успешные попытки такого диалога, который обычно заканчивается либо победой воинствующих лодырей благодаря агрессивной опеке их родителей, либо (чаще) иллюзией успешности и устраивающей всех профанацией. Всё это сильно понижает вероятность удержания в школе грамотного профессионала-программиста, обладающего педагогическим талантом.

С другой стороны, кадровые школьные учителя, проявляя недобросовестность, иногда даже в сильных школах и курсах хватаются за удобные им достаточно примитивные инструменты обучения —библиотеки, среды и языки программирования, при этом забывая о ежедневной борьбе за качество и архитектуру кода, за осмысленность деятельности, за неиллюзорное привитие навыков, которые помогут детям в больших проектах.

Они поступают, как им проще (а не как лучше), не удосуживаются следить за качеством кода обучаемых, за стилем и направлением их мышления, ограничиваясь лишь видимостью обучения. Такие образовательные иллюзии очень вредны. Заметны они становятся достаточно поздно, когда выясняется, что ученик, легко пишущий небольшие программы (пусть даже логически насыщенные, олимпиадные), принципиально не способен написать что-то большее, путается в коде, а другие, в том числе и профессионалы, его не понимают в силу спутанности его мышления и неумения внятно выразить мысли на уровне современных стандартов.

Это — типичнейшая картина на плоской равнине образовательных результатов современной средней школы по программированию. Чтобы преодолеть этот барьер, воздвигнутый нерадивым преподавателем (или нерадивостью ученика), приходится серьёзно и самостоятельно переучиваться — иногда будучи уже студентом или аспирантом. Либо смириться и «носить кофе программистам», чем все это достаточно часто заканчивается.

Искусство программирования — это искусство мышления, не надо это забывать, дорогие преподаватели и учащиеся .

Принципы, заложенные в TXLib для повышения качества обучения

Сделай сам. В **TXLib** многие вещи сделаны или оставлены не совсем удобными для применения. Это — предложение подумать, как сделать это самому, и, как правило, для этого в **TXLib** есть средства. Сделав, покажите решение другим, если они быстро поймут его и оценят — ваше решение удачное.

Загляни в Help. (Слово неспроста выбрано английским, потому что большинство информации в современном программировании — на английском языке. Учите его.) Под системой помощи понимается не только **TXLib Help**, но и весь Internet.

Посмотри, как сделано. Загляни в код библиотеки. Он создавался в том числе как пример программной системы со своей логикой и со своей реализацией, а некоторые функции можно понять только по коду, потому что их нет в системе помощи. Не всегда решения, применённые в **TXLib** оптимальны даже с точки зрения автора — он надеется, что это убережёт желающих обучиться качественно, но нетерпеливых учеников, от [Ctrl+C и Ctrl+V] плагиата.

Посмотри, как сделано иначе. **TXLib** — не единственная графическая библиотека, и реализация «простого графического холста», применённая в ней — не единственное решение. Посмотрите как устроены десятки других графических библиотек. Но избегайте плохого кода (его можно определить по тому, как морщатся профессионалы, глядя на него, если у вас нет более объективных средств такого определения) — он научит вас плохому. Хороший, но сложный код (глядя на него, профессионалы не морщатся, а начинают изучать) — отложите до времени и вернитесь к нему позже.

Выйди за пределы «песочницы». Это усиление принципа «сделай сам» — «собери вместе свои мысли про хорошую библиотеку, посмотри, как устроен **TXLib** и его аналоги, сделай свою библиотеку, лучше **TXLib**'а.» Примеры таких библиотек можно найти на сайте **TXLib** и в Интернете, и некоторые из них сделаны как раз начинающими.

Литература

1. Хант Э., Томас Д. Программист-прагматик. Путь от подмастерья к мастеру. – СПб, Питер, 2007. – 288 с.
2. Дединский И. Р. Как хотеть учиться. // Компьютерра. – 2005. – № 24.

Вводная документация к библиотеке TX Library

Библиотека Тупого Художника (The Dumb Artist Library, TX Library, TXLib)

Version: 00173a, Revision: 167

Date: 2020-06-02 20:23:03 +0400

Назначение

TX Library — компактная графическая библиотека для **Win32** на C++. Это небольшая «песочница» для начинающих реализована с целью помочь им в изучении простейших принципов программирования. Документация на русском языке.

Философия TX Library — облегчить первые шаги в программировании и подтолкнуть к творчеству и самостоятельности.

TX Library *is a tiny graphics library for* **Win32** *written in C++. It is a small sandbox for the very beginners to help them to learn basic programming principles. The documentation is currently in Russian.*

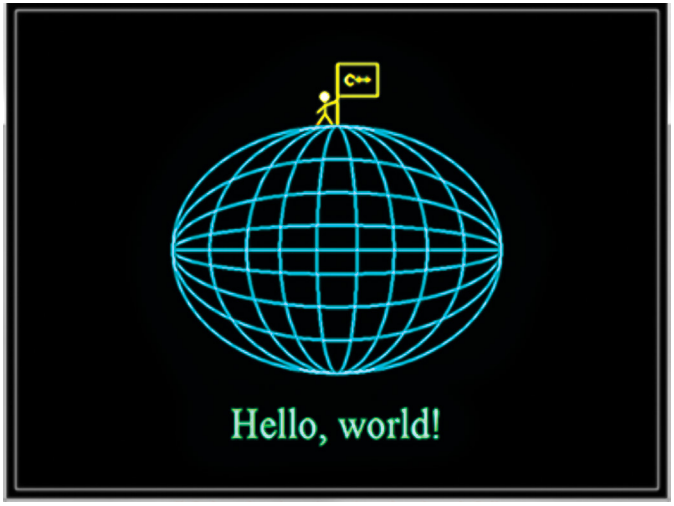
Официальный сайт библиотеки: txlib.ru.

См. также страницу проекта на SourceForge и GitHub. Короткая ссылка на онлайн-документацию: gg.gg/TXLib.

Аналогичный проект для **Linux/MacOS** (разработанный девятиклассником): txlin.roveramd.com.

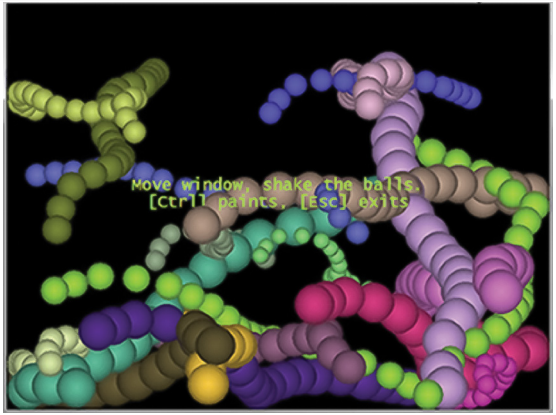
Правила использования материалов библиотеки и сайта см. на официальном сайте **TXLib**.

Скриншоты

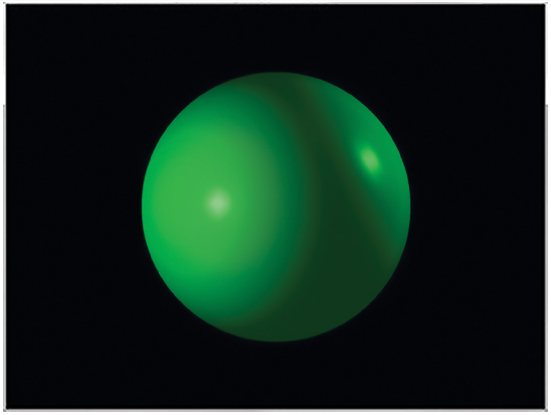


`Example03.cpp`: *Простейшая программа.*

`Movie.cpp`: *Мультфильм (автор — ученик 6 класса).*



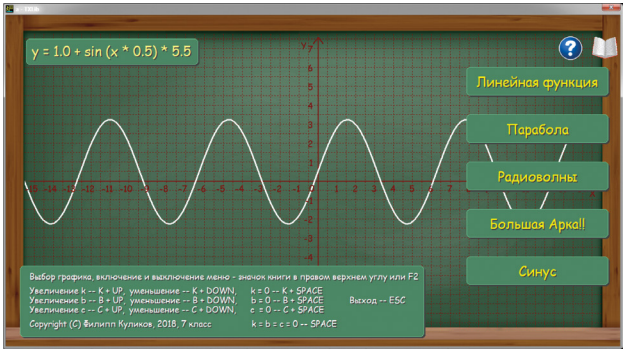
Shaker.cpp: *Простая физическая модель движения (автор — ученик 7 класса).*



Phong.cpp: *Модель освещения сферы (автор — ученик 9 класса).*

Tree.cpp: *Простая рекурсия: Дерево автор — ученик 7 класса).*

Tennis.cpp: *Использование картинок.*



`MathBeauty.cpp`: *Построитель графиков (автор — ученик 7 класса).*

Предупреждение, или TXLib – это всего лишь инструмент

Библиотека **TXLib** – это всего лишь инструмент для того, чтобы облегчить первые шаги в программировании. Однако этот инструмент, как и любой другой, может быть применен неправильно. (Тем не менее, в основу **TXLib** заложены некоторые принципы, помогающие конструктивному неиллюзорному обучению: квесты, отрицательные примеры, намеренная недостаточная и/или неудобная функциональность, мотивирующие подтрунивания и т.п.).

Сама по себе любая библиотека или язык программирования не научит начинающего писать программы грамотно. Научит этому разработка своих, достаточно больших проектов, в сочетании с тесным общением профессионалов, желающих помочь начинающим. Такие профессионалы должны обладать и опытом разработки больших программ, и педагогическими навыками, чтобы передать свой опыт начинающим.

К сожалению, не всегда это совпадает. Профессионалы-программисты зачастую не хотят лезть в обучение, где хватает своих проблем. С другой стороны, недобросовестные учителя, иногда даже в сильных школах и курсах, хватаются за удобные инструменты обучения (чужие или свои библиотеки, среды и языки программирования), не удосуживаясь следить за

качеством кода обучаемых, за стилем и направлением их мышления, ограничиваясь лишь видимостью обучения.

Такие образовательные иллюзии очень вредны. Заметны они становятся достаточно поздно, когда выясняется, что ученик, легко пишущий небольшие программы (пусть даже алгоритмически насыщенные, олимпиадные), принципиально не способен написать что-то большее, путается в коде, а другие, в том числе и профессионалы, его не понимают в силу спутанности его мышления и неумения внятно выразить мысли на уровне современных стандартов. Чтобы преодолеть этот барьер, воздвигнутый нерадивым преподавателем (или вашей собственной нерадивостью), приходится серьезно и самостоятельно переучиваться – иногда будучи уже студентом или аспирантом. Либо смириться и «носить кофе программистам».

Искусство программирования – это искусство мышления, не надо это забывать, дорогие преподаватели и учащиеся.

Принципы, заложенные в TXLib для повышения качества обучения:

Сделай сам. В **TXLib** многие вещи сделаны или оставлены не совсем удобными для применения. Это – предложение подумать, как сделать это самому, и, как правило, для этого в **TXLib** есть средства. Сделав, покажите решение другим, если они быстро поймут его и оценят, то ваше решение — удачное.

Загляни в Help. (Слово неспроста выбрано английским, потому что большинство информации в современном программировании – на английском языке. Учите его.) Под системой

помощи понимается не только **TXLib Help**, но и весь Internet.

Посмотри, как сделано. Загляни в код (см. «Исходные тексты»). Он создавался в том числе как пример программной системы со своей логикой и со своей реализацией, а некоторые функции можно понять только по коду, потому что их нет в системе помощи. Не всегда решения, примененные в **TXLib** оптимальны даже с точки зрения автора – он надеется, что это убережет желающих научиться качественно, но нетерпеливых учеников, от Ctrl+C и Ctrl+V плагиата.

Посмотри, как сделано иначе. **TXLib** – не единственная графическая библиотека, и реализация «простого графического холста», примененная в ней – не единственное решение. Посмотри, как устроены десятки других графических библиотек. Но избегай плохого кода (его можно определить по тому, как морщатся профессионалы, глядя на него, если у вас нет более объективных средств такого определения) – он научит вас плохому. Хороший, но сложный код (глядя на него, профессионалы не морщатся, а вздыхают) – отложи до времени и вернись к нему позже.

Выйди за пределы «песочницы». Это усиление принципа «сделай сам». Собери вместе свои мысли про хорошую библиотеку, посмотри, как устроен **TXLib** и его аналоги, сделай свою библиотеку, лучше **TXLib**'а. Это нетрудно. :) Примеры таких библиотек можно найти на сайте **TXLib** и в Интернете, и некоторые из них сделаны как раз начинающими.

Удачи, и May the Source be with you! :)

Установка библиотеки

Библиотека **TXLib** состоит из единственного файла и не требует никаких настроек в среде программирования, чтобы облегчить ее установку и работу для начинающих.

Скачайте программу установки, загрузка по ссылке начнется автоматически. Ее имя имеет вид `TXLib-v0173a.rar.exe`. Цифры могут отличаться (это номер версии), расширение .exe может не отображаться, в зависимости от текущих настроек **Windows**.

Запустите скачанную программу установки. Программа установки — это самораспаковывающийся архив, она не требует особых прав для запуска.

На рабочем столе появится «Ярлык для TX». Откройте его и запустите систему помощи **TXLib Help**, изучите её. Примеры см. в папке Examples, в папке Examples/Demo.

Если при установке происходят ошибки или запуск программы установки невозможен, откройте файл библиотеки TXLib.h отсюда, сохраните (Ctrl+S) его в свою рабочую папку, где вы сохраняете свои программы. Пользуйтесь системой помощи онлайн.

Для полной обработки ошибок библиотеке требуются модули, которые желательно установить (скопировать) в папку Windows. Устанавливать эти библиотеки не обязательно. Программы, использующие TXLib, будут запускаться и без них.

Модули библиотеки Microsoft DBGHELP для доступа к отладочным символам **Microsoft**:

`dbghelp32.dll` для 32-разрядных программ (либо `dbghelp.dll`, 32-разрядная версия),

dbghelp64.dll для 64-разрядных программ (либо **dbghelp.dll**, 64-разрядная версия).

Модули библиотеки DrMinGW для доступа к отладочным символам **MinGW** компилятора **GCC g++:**

mgwhelp32.dll для 32-разрядных программ (либо **mgwhelp.dll**, 32-разрядная версия),

mgwhelp64.dll для 64-разрядных программ (либо **mgwhelp.dll**, 64-разрядная версия).

Суффиксы 32 и 64 помогают отличить 32-разрядную и 64-разрядную версии DLL-файлов библиотек. Например, **dbghelp32.dll** — это просто переименованная 32-разрядная версия файла **dbghelp.dll**.

Для наиболее полной диагностики ошибок полностью отключайте оптимизацию при компиляции. Например, для компилятора **GCC g++** — с помощью ключа командной строки **-O0**. Разные среды программирования позволяют задать эти ключи по-разному, например, в **CodeBlocks** через **Главное меню — Settings — Compiler — (Global Compiler Settings) — (Compiler Settings) — Other Options**.

Кодовая страница в редакторе среды разработки должна быть установлена как **Windows CP1251**, проверьте это. В разных средах разработки она устанавливается по-разному, например, в **CodeBlocks** через **Главное меню — Settings — Editor — (General Settings) — Other Settings — Encoding.** Иначе русские буквы в сообщениях **TXLib** будут отображаться неправильно.

Особенности использования

Файл TXLib.h должен быть включен (#include) в программу:

До или вместо файла windows.h. Либо надо задать (#define) константы WINVER и _WIN32_IE не ниже 0x0500 и отменить (#undef) макросы UNICODE и _UNICODE.

До или вместо файлов string.h или stdio.h, если используется режим строгого соответствия стандарту ANSI. Имейте в виду, что эти файлы могут включаться в windows.h.

После включения библиотек boost, использующих модуль boost::preprocessor.

При использовании вместе с SFML необходимо перед включением (#include) в программу задать макрос TX_USE_SFML (#define TX_USE_SFML). **TXLib** по умолчанию включает режим отладки стандартной библиотеки glibc, а SFML, как правило, собрана без этого режима, что приводит к ложным сообщениям об ошибках, связанным с динамической памятью, например при использовании класса sf::Text.

Файл библиотеки большой и может компилироваться долго, поэтому обратите внимание на возможность использования прекомпилированной версии в проектах с раздельной компиляцией. См. макрос TX_COMPILED. Также можно определить макрос WIN32_LEAN_AND_MEAN до включения TXLib.h в программу.

Поддерживаемые платформы

Среды программирования и исполнения + компиляторы

MinGW GCC 9.2.0 (NUWEN 64-bit, TDM-GCC 32-bit и 64-bit)
MinGW GCC 8.2.0 (NUWEN 64-bit, TDM-GCC и MinGW.org 32-bit и 64-bit)
MinGW GCC 7.2.0 (NUWEN 64-bit, TDM-GCC 32-bit и 64-bit)
MinGW GCC 6.3.0 (NUWEN 64-bit, TDM-GCC 32-bit и 64-bit)
MinGW GCC 5.3.0 (NUWEN и TDM-GCC, 32-bit и 64-bit)
CodeBlocks 20.03 + MinGW GCC 8.1.0 (MinGW.org, 32-bit и 64-bit)
CodeBlocks 12.11 + MinGW GCC 4.8.0 (NUWEN)
CodeBlocks 12.11 + MinGW GCC 4.7.2 (NUWEN)
Dev-CPP 5.11 + MinGW GCC 4.9.2 (TDM)
Qt Creator 4.2.2 + MinGW GCC 5.3.0 (Qt), 32-bit (кодировка файлов в редакторе: только CP1251)
Cygwin + Cygwin GCC 7.4.0, 32-bit и 64-bit
LLVM + MSVC Clang-cl 9.0.0, 32-bit и 64-bit
Microsoft Visual Studio 16 (2019), 32-bit и 64-bit
Microsoft Visual Studio 15 (2017), 32-bit и 64-bit
Microsoft Visual Studio 14 (2015), 32-bit и 64-bit
Microsoft Visual Studio 12 (2013), 32-bit и 64-bit
Microsoft Visual Studio 11 (2012), 32-bit и 64-bit
Microsoft Visual Studio 10 (2010), 32-bit и 64-bit
Intel C++ Compiler 13.0 (2013)

Операционные системы

```
Windows 10 ru/en, 32-bit и 64-bit
Windows 8 ru/en, 32-bit и 64-bit
Windows 7 ru/en, 32-bit и 64-bit
Windows Vista ru/en
Windows 2003 SP1 R1/R2 ru/en
Windows XP SP2/SP3 ru/en
Windows 2000 SP4 ru/en
Linux + Wine
Ubuntu 16.04, 11.04 + Wine
MacOS X + Wine
Snow Leopard 10.6.8 + Wine
```

Заметки:

Принудительно отключается режим `UNICODE` (кроме редактора ресурсов **Visual Studio**). Для явной работы с `UNICODE`-функциями используйте суффикс `W`.

Принудительно включается режим:

`_CRT_SECURE_CPP_OVERLOAD_SECURE_NAMES`.

Предупреждения:

Если программа разрабатывалась на определенных платформах с настройками по умолчанию, то во время исполнения могут потребоваться файлы библиотек, без которых программа не запустится. Посмотреть эти зависимости можно утилитой `Depends.exe`.

Простейшая программа

Простейшая программа на C (или C++) состоит из двух частей: раздела подключения библиотек и главной функции программы. Рассмотрим пример, в котором рисуется на экране простой рисунок.

Раздел подключения библиотек

Директивы (команды) подключения библиотек находятся обычно в начале файла программы и выглядят обычно так:

```
#include <stdlib.h>

#include "TXLib.h"
```

`Stdlib.h` и `TXLib.h` — файлы библиотек, которые подключаются к вашей программе. После этого вы можете использовать команды, функции и переменные, объявленные в этих файлах. Больше, чем нужно, библиотек подключать не стоит, хотя это и не вредно. Когда используется много библиотек, раздел может быть очень большим.

Главная функция программы

Программа на C (C++) состоит из функций. Функция — это описание каких-либо действий с заданным именем (названием).

Например,

```
int main()
    {
    txCreateWindow (800, 600);
    txLine (320, 290, 320, 220);
    return 0;
    }
```

Главная функция — это функция, с которой начинается исполнение программы. Её имя — `main()`. Скобки показывают, что речь идет именно о функции, т.е., об описании каких-то действий. Для того, чтобы функция начала работу, её нужно вызвать. Функцию `main()` вызывает сама операционная система компьютера. Слово `int` означает, что `main()` в конце работы передаст тому, кто ее вызывал (операционной системе), некое целое число. Это число для функции `main()` означает код завершения нашей программы. Если он равен `0`, то работа программы считается успешной.

Действия, записанные в функции, заключаются в фигурные скобки { и }. Они обозначают начало и конец функции.

Внутри функции записаны вызовы команд, которые что-то рисуют на экране. Между командами, там, где это логически необходимо, стоят пустые строки, отделяющие одни части программы от других. Это способствует большей понятности программы. Пустая строка в программировании соответствует началу абзаца текста в русском языке.

Для понимания программы и того, чтобы в ней не появлялись ошибки, очень важно, чтобы в нужных местах в ней стояли пробелы. Обычно их ставят до открывающих круглых скобок, после запятых, до и после знаков операций. Наличие пробелов делает программу приятной на вид, и предотвращает напряжение глаз при работе с компьютером. Работа с плохо оформленным текстом программы может нанести вред глазам, снизить зрение.

Пример плохо написанной программы (так писать НЕ надо):

```
#include «TXLib.h»
 int   main(){
 txCreateWindow(800,600);
 txLine(320,290,320,220);
  txLine (320, 290,280,  350);
    txLine(320, 290,360,350);
  txLine(320,230,270,275);
txLine(320,230, 400,   220);
 txCircle(320,190,30);
   txSelectFont( “Times”,60);
 txTextOut(240,400,”Hello, world!”);
          return        0;
 }
```

```
FFFFF  FFFFF  FFFFF  UU  UU  UU  UU  UU  UU  UU  UU  !!  !! !!
FF     FF     FF     UU  UU  UU  UU  UU  UU  UU  UU  !!  !! !!
FF     FF     FF     UU  UU  UU  UU  UU  UU  UU  UU  !!  !! !!
FFFFF  FFFFF  FFFFF  UU  UU  UU  UU  UU  UU  UU  UU  !!  !! !!
FF     FF     FF     UU  UU  UU  UU  UU  UU  UU  UU  !!  !! !!
FF     FF     FF     UU  UU  UU  UU  UU  UU  UU  UU  !!  !! !!
FF     FF     FF     UU  UU  UU  UU  UU  UU  UU  UU
FF     FF     FF      UUUU    UUUU    UUUU    UUUU   !!  !! !!
```

Очевидно, что таким стилем программирования зрение (а также и мозги) будет быстро и безвозвратно испорчено. **:(**

Для задания положения рисуемых фигур используется координатная система, у которой начало координат расположено слева-вверху, а ось **OY** смотрит вниз. Это несколько непривычно, но так традиционно принято в компьютерной графике, поэтому деваться некуда. :) Например, команда

```
txLine (320, 290, 320, 220);
```

проводит линию из точки **x=320** и **y=290** в точку с **x=320** и y=**220**.

Каждая команда заканчивается точкой с запятой. Это — аналог точки в русском языке. Точка с запятой ставится в конце каждого законченного действия или объявления. Например, в строке с `txCreateWindow (800, 600)` точка с запятой ставится, т.к. в этой строке «закончена мысль» о том, что надо создать окно для рисования. В строке с `int main()` — не ставится, т.к. описание функции `main()` не закончено (на самом деле, оно там только начато).

Если в программе используются строки, они заключаются в двойные кавычки, например:

```
txTextOut (240, 400, «Hello, world!»);
```

Если в программе присутствуют числа с дробной частью, то эта часть отделяется от целой части точкой, а не запятой, как в русском языке.

Распространенные ошибки и борьба с ними

Первое правило в борьбе с ошибками: не бойтесь их количества. Часто первые ошибки влекут за собой последующие. Поэтому прежде всего анализируйте и исправляйте самую первую ошибку. Остальные могут быть её следствием и исправятся сами, если вы исправите первую ошибку.

Второе правило: будьте внимательны. Об этом *см. ниже*.

Третье правило волшебника: ищите ошибку вначале у себя, а только потом в библиотеках. Библиотеки достаточно долго тестировались, поэтому вероятность ошибки в них очень мала.

Ошибки, не связанные с текстом программы

Вначале убедитесь, что файл с программой имеет правильное расширение «`.CPP`», а не расширение «`.C`». Если расширение (тип) файла неправильное, то в списке ошибок одной из первых будет сообщение «`TXLib.h: Must use C++ to compile TXLib.h`». Это потому, что для библиотеки требуется компилятор **C++**, а он транслирует файлы с расширением «`.CPP`». Файлы «`.C`» транслирует компилятор **Си**, а не **C++**. Чтобы сменить расширение файла, выберите в главном меню `[File]`, `[Save As]` и дайте программе имя с расширением «`.CPP`». Лучше не использовать ни русских букв, ни пробелов в имени файла.

Если файл библиотека **TXLib** установилась неверно, то файл `TXLib.h` может быть не найден. Сообщение об ошибке зависит от компилятора, но будет содержать фразу, подобную «`TXLib.h`

file not found». В этом случае выйдите из среды программирования и переустановите **TXLib**. Если это не помогает, то просто скопируйте файл TXLib.h из папки **TX** в папку с примерами или в папку с вашей программой.

Ошибки, связанные с текстом программы

В простых программах ошибки бывают двух видов:

Ошибки записи (орфография, пунктуация) — их называют синтаксическими ошибками (**syntax error**). Они происходят до запуска программы, на стадии перевода программы в машинный код (стадии компиляции). Поэтому их называют ошибками времени компиляции (**compile-time errors**). Исполняемый файл при этом не создается и программа не запускается.

Логические ошибки — они происходят после запуска программы, при этом при компиляции ошибок нет (иногда имеются предупреждения, warnings, которые полезно исправлять, а лучше не допускать их появления). Их называют ошибками времени исполнения (**runtime errors**).

Ошибки времени компиляции

Практически все синтаксические ошибки на этой стадии происходят из-за невнимательности.

Распространённые синтаксические ошибки:

Путают ключевые слова, названия библиотек и команд:

```
#iclude “TX lib.h”        Правильно: #include “TXLib.h”
in maim()                 Правильно: int main()
tx Line (10, 10, 20, 20)  Правильно: txLine (10,10,20,20);
```

Путают большие и маленькие буквы:

```
txcircle (100, 100, 10)
```
Правильно: `txCircle`

Не ставят скобки:

```
int main
```
Правильно: `int main()`

Забывают запятые или путают их с другими знаками:

```
txCircle (100  100  10)
```
Правильно: `txCircle (100,100,10);`

```
txCircle (200; 200; 20)
```
Правильно: `txCircle (200,200,20);`

Забывают точки с запятой:

```
txSelectFont (“Times”,60)
```
Правильно: `txSelectFont (“Times”,60);`

Забывают или путают кавычки:

```
txSelectFont (Times, 60)
```
Правильно: `txSelectFont (“Times”, 60);`

```
txSelectFont (‘Arial’, 20)
```
Правильно: `txSelectFont (“Arial”, 20);`

Ставят лишние точки с запятой там, где «мысль не закончена»:

```
int main();
```
Правильно: `int main()`

Указывают дробную часть числа не через точку, а через запятую:

```
3,1415
```
Правильно: `3.1415`

Забывают фигурные скобки:

```
int main()
```
Правильно:

```
int main()                          {
txCreateWindow (800, 600);         txCreateWindow (800, 600);
txLine (320, 290, 320, 220);      txLine (320, 290, 320, 220);
return 0;                           return 0;
                                    }
```

Забывают писать открывающие или закрывающие скобки, или пишут лишние, отчего появляются непарные скобки, или путают виды скобок:

```
int main()
    {
    txCreateWindow (800, 600);
    txLine (320, 290, 320, 220; // Ошибка: Не закрыта круглая скобка
    return 0;
    }
```

```
int main()
    {
    txCreateWindow {800, 600};  // Ошибка: Фигурные скобки вместо круглых
    return 0;
    }
```

```
  txLine (320, 290, 320, 220); //Ошибка: Команда за пределами функции main
    }                                 //Ошибка: Лишняя скобка
```

Помещают команды за пределы фигурных скобок

```
int main()
    {
    txCreateWindow (800, 600);
    return 0;
    }
txLine (320, 290, 320, 220); // Ошибка: Команда за пределами функции main()
```

Указывают лишние данные в командах или указывают меньше данных (аргументов), чем нужно.

При таком несоответствии количества параметров (**too few arguments...** — слишком мало агрументов, или **too many arguments...** — слишком много) среда программирования часто указывает на правильное определение той команды, которая была неверно вызвана. Это нужно для того, чтобы вы посмотрели на это определение и вспомнили, сколько данных надо передавать. Это совсем не означает «ошибки в библиотеке» или «ошибки в стандартной команде». Настоящее место ошибки там, где команда вызвана. В это место легко попасть, если кликнуть мышкой 2 раза на строку с надписью «**...at this point in file**» в списке ошибок, в нижней части окна среды программирования.

Примеры таких ошибок:

```
txCircle (100, 100);            //Здесь не указан радиус
txCircle (200, 200, 20, 30);    //Здесь лишнее число: txCircle() принимает всего 3 аргумента
```

Ошибки времени исполнения

Бывают и логические ошибки, или ошибки времени выполнения (runtime errors). Например, мы не выбрали нужный цвет рисования (по умолчанию он белый), или цвет совпал с фоном и поэтому не виден. Или мы нарисовали одну фигуру поверх другой, и она закрыла предыдущую, или задали неверные координаты. Эти ошибки появляются после запуска программы. Их легко понять, если выполнить на листе бумаги все команды одну за другой, в том порядке, в котором они заданы в программе. При этом не надо стараться выполнять «так, как лучше», или «как хочется, чтобы получилось». Выполняйте так, как будто это не ваша, а совсем чужая работа, и вам не интересен её результат. В тот момент, когда результат разойдётся с вашим желанием и вы получите странную фигуру, станет понятно место ошибки.

Конечно лучше, если ошибка произошла на стадии компиляции, чем на стадии выполнения, когда искать её труднее. Поскольку человеку избежать ошибок невозможно, один из видов мастерства состоит в том, чтобы постараться перевести хотя бы часть ошибок из стадии исполнения на стадию компиляции. Пусть они происходят там, где их легче ловить. :)

Ошибки и стиль программирования

Правильное оформление кода (стиль программирования) помогает перевести ошибки стадии выполнения на стадию набора текста, и находить их даже до компиляции. Например, неверные координаты очень сложно проследить в «примере плохо написанной программы» (см. выше). Сделать это гораздо легче в «законченном примере» (см. ниже), где все числа, операторы и команды аккуратно выровнены. Если программа длиннее нескольких строк,

то между её логическими частями ставят пустые строки (вместо «красных строк» в русском языке), это позволяет не напрягать логическое мышление зазря, а концентрироваться на творчестве и на развитии программы.

Оформлять код «красиво» (а на самом деле ясно и логично) уже после того, как программа написана и отлажена, достаточно бессмысленно — хороший стиль облегчает написание и отладку, а ведь программа уже готова. :) Поэтому ставить пробелы, пустые строки и выравнивать текст нужно сразу, при наборе, и приучиться это делать автоматически, как (хочется надеяться) вы автоматически моете руки перед едой. :).

Кто ясно мыслит — ясно излагает. Кто ясно излагает и пишет — приучается ясно мыслить, письменная речь и мышление тесно связаны и, по общепринятой концепции Хомского, обуславливают друг друга.

Многие опытные люди, желая помочь вам, бывают разочарованы плохим стилем и перестают помогать и советовать. Поэтому чтобы регулярно получать помощь знатоков, нужно держать стиль на высоте. К счастью, оформить программу — это самое лёгкое дело в программировании, но и одновременно это первый шаг к надёжности и мастерству.

Если вам непонятно, что означает та или иная команда в программе, посмотрите в системе помощи **TXLib Help** «Простейший пример» или помощь по этой команде в разделе «Рисование». Например, по функциям `txLine()` или `txTextOut()`.

Полную версию системы помощи см. на сайте: http://txlib.ru.

Законченный пример

Текст программы | Примечания

```
#include «TXLib.h»                                  //Подключение библиотеки рисования

int main()                                          //Заголовок главной функции
  {                                                 //Начало функции
  txCreateWindow (800, 600);                      //Открытие окна размером 800 на 600 пикселей

  txLine (320, 290, 320, 220);  //Проводится линия из точки x=320, y=290 в точку с x=320, y=220

  txLine   (320, 290, 280, 350);                    //Проводятся другие линии,
  txLine   (320, 290, 360, 350);                    //в результате чего
  txLine   (320, 230, 270, 275);                    //на экране
  txLine   (320, 230, 400, 220);                    //появляется рисунок
  txCircle (320, 190, 30);       //Рисуется окружность с центром x=320 y=190 и радиусом 30

  txSelectFont (”Times New Roman”,60);              //Выбирается шрифт «Times» размера 60
  txTextOut (240, 400, ”Hello, world!”);  //Печатается текст «Hello, world!» в точке x=240 y=400

  return 0;                                 //Возврат успешного кода завершения (0 = нет ошибок)
  }                                                 //Конец функции
```