Редкая профессия

> **Комментарий 2008 года**: сокращенный вариант статьи был опубликован в виде отдельной статьи в декабрьском номере журнала PC Magazine/Russian Edition за **1997** год. Статья до недавнего времени находилась в online-архиве журнала, однако была удалена (очевидно, в связи с истечением срока давности ☺).

Недавно поздно вечером к нам в комнату зашел коллега, хороший знакомый, глава маленькой фирмы, "широко известной в узких кругах", устало опустился на стул и, очумело покрутив головой, проговорил:

- От заказчиков отбоя нет!..

Он поднял голову, и мы увидели в его усталых глазах удивление, смешанное с восторгом и воодушевлением.

- Работы полно, только успевай!

Его фирма в принципе ничем не отличается от других московских компьютерных фирм, ориентирующихся на программно-аппаратные разработки, - несколько постоянных сотрудников, десяток толковых студентов, три комнаты в умирающем академическом институте и минимальный, мягко говоря, набор оборудования.

Я вспомнил этот случай, чтобы показать безосновательность сетований на упадок индустрии, невостребованность специалистов и повсеместную ориентацию на продажу готовых западных решений. Сегодня можно уверенно говорить, что неглупый инициативный компьютерщик - от студента до сорокалетнего программиста со стажем - имеет все возможности найти себе работу, соответствующую его квалификации, за вполне адекватные деньги.

Речь, однако, о другом. Разработка программного обеспечения - настолько широкая область деятельности, что человек, который действительно хорошо программирует, скажем, драйверы устройств и зарабатывает этим себе на жизнь, вряд ли сможет в приемлемый срок научиться профессионально проектировать базы данных. Исключения крайне редки. (Я не беру в расчет одержимых молодых людей, которые имеют свое мнение решительно обо всех аспектах разработки и использования ПО и пишут в своих резюме невообразимо длинный список программных систем самого разного калибра и назначения, в которых они как бы умеют работать. Речь идет прежде всего о настоящих профессионалах.) В то же время, и драйверы, и базы данных определенно относятся к программному обеспечению.

Жизнь сложилась так, что автор любит проектировать и разрабатывать компиляторы, и у него это, в общем, получается. То, что вы прочтете ниже, надеемся, подтвердит мои слова. Однако чем серьезнее относишься к тому делу, которое любишь, тем меньше времени и возможностей остается постичь что-то другое, пусть даже важное. Боюсь, ни за какие деньги я не возьмусь сейчас, скажем, за графику или разработку бухгалтерских программ - не только потому, что мне это не интересно, но прежде всего из-за невозможности достаточно быстро стать профессионалом в этих сферах.

А теперь скажите, какая софтверная специализация пользуется сейчас на рынке большим спросом: разработка компиляторов или компьютеризация бухгалтерий? Ответ очевиден. Более того, возможно, такая же картина наблюдается и на Западе. В самом деле, даже если там существует *несколько сотен* аттестованных компиляторов языка Ада (а это именно так), количество специалистов, их программировавших, все равно значительно меньше числа тех, кто разрабатывает всевозможные прикладные программы по заказам фирм и организаций.

> **Комментарий 2002 года**. На Западе, с его несоизмеримо б**о**льшим разнообразием потребностей в сфере программирования, ситуация все-таки иная. Когда происходили описываемые ниже события (середина 90-х), я, скажем, не знал о существовании web-сайта http://www.compilerjobs.com/, в котором публикуется и регулярно обновляется поразительный в своем разнообразии список вакансий, связанных с разработкой компиляторов…

Так что заголовок этой части имеет вовсе не хвастливый, а грустный оттенок. Разработка компиляторов - *редкая* профессия, но не в смысле ее дефицитности и, следовательно, высокой цены на нее. Напротив, она редкая прежде всего потому, что редко требуется и, соответственно, найти себе применение крайне трудно. Поэтому, когда все-таки появляется работодатель, предлагающий такую работу, психологически трудно еще и педантично уточнять условия вроде денег и прочего: спасибо и на том, что работа, по-настоящему интересная и приносящая радость, нужна кому-то еще!

Летает или не летает?

Я хочу рассказать о том, как мы делали компилятор Си++. Вообще-то, об этом стоило бы написать книгу - настолько эта история кажется захватывающей и поучительной, однако... будет ли это кому-нибудь интересно? Даже если коротко рассказать о наиболее существенных проблемах, с которыми мы столкнулись, трудно избавиться от мысли о, скажем, неактуальности нашего опыта в сегодняшней российской ситуации в программировании. В самом деле, посмотрите хотя бы на полки отделов книжных магазинов, торгующих компьютерной литературой. Невероятное (по сравнению с картиной 5-6-летней давности) разнообразие книг! Практически по любому программному продукту, мало-мальски используемому у нас, можно гарантированно найти по крайней мере две-три книги. Однако работ, посвященных современным архитектурам, проблемам разработки программного обеспечения, принципам построения сложных систем, таких, как компиляторы, СУБД, операционные системы, - нет. Только описания конкретных инструментов, пакетов и систем. Ситуация в некотором смысле обратная той, которая складывалась в доперестроечное время: тогда очень многие серьезные работы известных западных авторов, пусть с опозданием на пару лет, но выходили у нас. Печатались и очень неплохие отечественные книги. (И между прочим, находили спрос, и многие мгновенно становились библиографической редкостью!)

Все это определенно говорит о том, что спрос на подобного рода публикации практически отсутствует. Кому сейчас интересна проблематика разработки компиляторов, когда у любого программиста на выбор имеются три-четыре превосходных западных продукта, а наша промышленность уже давно не производит собственных компьютеров, для которых могли бы понадобиться подобные разработки?

У этих опасений есть еще один аспект. Опыт общения с западными специалистами по ПО, как достаточно известными и уважаемыми, так и рядовыми программистами, убедил в одной простой вещи: практически никому не интересны те проблемы и трудности, с которыми ты сталкиваешься в процессе реализации того или иного проекта; мало кого интересуют пути и способы их преодоления. Важен и интересен прежде всего результат! Об этом образно сказал нам один коллега, эмигрировавший в Швейцарию лет пятнадцать назад.

- Вот *ваш* "Буран", - сказал он (дело было больше двух лет назад), удобно расположившись в кресле на открытой террасе Политехнического института в Лозанне с видом на Женевское озеро.- Программа вроде бы завершилась единственным полетом в беспилотном режиме. Наверняка в процессе его разработки конструкторы продемонстрировали высокую квалификацию, нашли какие-то интересные нестандартные решения, придумали и отработали технологию, решили уйму проблем и т.д. и т.д. Но... - заключил он с ехидцей в голосе,- *не летает*! Не делает то, для чего был предназначен! А значит, и говорить о нем бессмысленно. Его нет, и это главное.

Светило ласковое солнце. Сквозь большие окна факультета информатики был виден просторный студенческий компьютерный класс, уставленный огромными цветными мониторами Sparc'ов. В сырой и холодной Москве слетавший в космос «Буран» сиротливо пристроился в парке Горького среди аттракционов. Возразить было нечего.

С тех пор в наших разговорах метафора "летает - не летает" приобрела в применении к нашему проекту почти ритуальный характер. Она не потеряла свою актуальность даже и сейчас, когда вроде бы по результатам тестирования на соответствие стандарту Си++ наш компилятор стабильно обгоняет все последние версии Watcom и транслирует сам себя почти так же быстро, как это делает Visual C++.

Компилятор "не летает". То есть не распространяется, не используется, не применяется в конкретных разработках. Почему - отдельная история.

Что нам стоит дом построить?

Однажды (около четырех лет назад) в НИИ при Московском университете, где мы тогда работали, появился высокий представительный мужчина с вальяжными манерами, окладистой бородой, большой лысиной и выразительными глазами навыкате.

Ничего удивительного в этом не было. Иностранцы посещали нашу "контору" через день, а то и чаще; каждая лаборатория либо выполняла какие-нибудь работы по западным заказам, либо изо всех сил старалась их получить. Это было тяжелое время: реальная зарплата истончалась с каждым месяцем, работы и каких-либо перспектив совершенно не было, немногочисленные предложения от государственных организаций носили отчетливый оттенок идиотизма и сиюминутности и подкреплялись безумно мизерным финансированием. Большие ЭВМ, когда-то работавшие круглосуточно и обслуживавшие весь университет, стояли; некоторым уже был подписан смертный приговор - золотосодержащие детали оказались более привлекательными, чем машинное время... НИИ медленно умирал и потихоньку пустел.

Каждому приходилось самому искать дополнительный приработок. Правда, возможности были. Преподавали в многочисленных тогда учебных центрах

совместных предприятий, писали для них учебные пособия, а то и книги, подрабатывали в коммерческих фирмах. Иностранные представители, которым наш директор с гордостью показывал "компьютерный класс" с десятком тайваньских XT, вежливо выслушивали его объяснения, сдержанно кивали и фотографировали со вспышкой, словно доисторическое чудовище, карточный перфоратор Juki, стыдливо задвинутый в дальний угол. Их предложения о совместных проектах (по крайней мере, доходившие до нашего отдела) либо носили несколько авантюрный и несолидный характер, либо были откровенно неинтересны. Несмотря на трудности и безвременье, мы все-таки ощущали себя системными программистами, и как-то не очень хотелось заниматься рисованием окон и конструированием экранных форм или переводом математических библиотек с одного языка программирования на другой. К тому же деньги предлагались оскорбительно скромные.

Однако то, что говорил высокий солидный бородач (назовем его Вальтер Деккер), звучало как откровение. Предлагалось разработать (не адаптировать, не доделать, не участвовать в разработке, а самим сделать from scratch - с нуля!) компилятор (компилятор!) с языка Си++ (!) для одной европейской (для определенности пусть для бельгийской) софтверной компании! Причем не какой-нибудь препроцессор в Си, как известный cfront, а честный прямой компилятор переднего плана, генерирующий низкоуровневый промежуточный код, используемый фирмой в системе программирования, в составе которой компиляторы Си, Модула-2 и Фортран.

Следует объяснить, что автор со студенческих лет питает к проблематике компиляции языков программирования особую страсть и имел к тому времени некоторый опыт как в проектировании языков (в частности, языков дискретного моделирования), так и в реализации различных языковых систем, включая (страшно сказать) системы построения компиляторов. Этот опыт целиком относился к прежним временам, а проекты, за немногими исключениями, носили полуинициативный характер, будучи поддержанными только непосредственным начальством. Проекты не мешали текущей работе в "ящике" (правильнее сказать, текущая работа не слишком препятствовала этим проектам). Поэтому предложения бельгийца казались невероятной удачей и в то же время справедливой наградой за долгие годы верности избранному направлению.

Фирма хотя и не обладала именем, звучащим в мировом масштабе, но казалась вполне респектабельной: она существовала уже более 25 лет, что для софтверной фирмы, согласитесь, немало, участвовала в нескольких общеевропейских проектах; ее продукты (в том числе, собственная коммерческая реализация UNIX) имели не одну тысячу пользователей. Так что желание дополнить свою систему программирования новым мощным языком выглядело вполне логичным.

> **Комментарий**. Любителям разгадывать псевдонимы и умолчания сказанного вполне достаточно, чтобы узнать страну, компанию, да и ее представителя.

Первый настораживающий момент (хотя то, что нам следовало тогда насторожиться, мы поняли гораздо позже) прозвучал вскоре после начала переговоров. Когда речь зашла о составе команды и предполагаемых сроках, то шеф, профессор Владимир Александрович Сухомлин, сам имеющий высокую квалификацию и немалый опыт подобных работ, не менее нашего опьяненный перспективой настоящего дела, немедленно ответил: три разработчика за один год. Сейчас мы понимаем, что здесь насторожиться следовало бельгийцам, уж

конечно, прекрасно знающим, каковы реальные трудозатраты подобных разработок: не имеют ли они дело с неопытными авантюристами? Однако они просто спросили: не очень мало? Тогда шеф, сделав для солидности паузу, сказал: ну ладно, год и четыре месяца.

Однажды в эхо-конференции по языку Ада - comp.lang.ada - прозвучал вопрос от некоего молодого человека по имени Майк Уайт. Этот замечательный парень из Массачусетса написал примерно следующее: вот на Макинтошах нет приличного компилятора для новой редакции Ады, так, может, я бы его сделал? Сколько примерно это заняло бы времени? Вопрос звучал слишком наивно, да и имя сильно смахивало на псевдоним, так что это вполне можно было бы принять за провокацию. Правда, говорят, американцы вообще довольно простодушный народ...

Что тут поднялось! Крупнейшие специалисты по языку Ада, мировые знаменитости вроде Роберта Девара всем своим весом (кто его видел, тот поймет мою иронию) обрушились на бедного Майка. "Вы сумасшедший! Вы не представляете, что такое сделать компилятор! Вы плохо изучали в университете курс по компиляции языков! Вы никогда не доведете этот проект до конца! На это требуется минимум 25-30 человеко-лет!" Тот, кажется, несколько ошарашенный этим тайфуном, растерянно отписывался: "Да... теперь я понимаю... это невозможно... лучше портировать GNAT на Макинтош... А может, мы с кем-нибудь скооперируемся и вместе все-таки попробуем?.."

Как знать, если бы этот американский Миша Белов не наткнулся тогда на столь суровую и дружную отповедь, быть может, он сейчас с парой приятелей уже заканчивал бы свой компилятор? Хорошо известно, что очень многие достойные проекты (примеры известны всем) выполнялись предельно малыми силами. И если бы мы, подобно Майку, перед тем как начать работу, спросили бы в comp.lang.cpp: друзья, а получится у нас компилятор - втроем за год?- почти наверняка получили бы аналогичный шквал критики.

Тогда мы об этом не думали. Конечно, мы знали Си++ только как пользователи, сам язык еще не приобрел своей теперешней монструозности, да и работа казалась настолько заманчиво-интересной и в то же время ясной, что инстинктивно хотелось заинтересовать собой фирмачей, не оттолкнув их слишком большими сроками. Но они-то, они - сделавшие и UNIX, и серию компиляторов, замахивающиеся на еще более амбициозные проекты, казалось, собаку съевшие на управлении программными разработками,- как они могли не насторожиться?

Они не удивились. Они сказали: "Хорошо, пишите план на полтора года".

По рукам!

У кого-то из классиков есть забавная шутка (за точность цитаты не ручаюсь, но смысл передан верно): "Системные программисты не вполне понимают, за что им платят большую зарплату: ведь они выполняли бы свою работу и бесплатно. Правда, у них хватает ума не говорить об этом своему начальству". Это очень точное наблюдение - прямо про нас, только без "большой зарплаты".

Характер условий, предлагаемых бельгийцами, был, видимо, типичен для тогдашних отношений с иностранными фирмами: они передавали нам несколько рабочих станций Sun 3/60 на процессоре Motorola 68020 (которые к тому времени уже морально устарели и самой фирмой попросту списывались) и обещали платить... не скажу сколько, но, поверьте, очень и очень небольшие деньги - даже по тогдашним российским меркам.

Вообще-то, это не очень вязалось с обликом солидной фирмы, но осознание пришло гораздо позже. Да, еще: те станции, на которых мы должны были работать, передавались нам не просто так - они шли в счет оплаты за нашу работу! Мы оказались как бы должны им, еще не приступив к делу... Впоследствии то же произошло со SparcClassic. Когда на 3/60 стало совсем невозможно работать (примерно как если на XT пытаться редактировать графические изображения), нам перевели деньги на покупку этой самой младшей модели семейства Sparc с минимальной комплектацией, внеся ее стоимость в оплату проекта.

Естественно, результат работы становился собственностью фирмы, а с нас взяли подписку о неразглашении, согласно которой в течение всего срока действия контракта мы не имели права не только говорить об условиях работы и названии фирмы, но и вообще рассказывать о том, что мы делаем. Говорят, некоторые компании оговаривают и более суровые условия: скажем, после окончания действия контракта разработчик в течение некоторого времени не имеет права работать над аналогичными проектами. Какое счастье, что бельгийцы до этого не додумались - мы бы подмахнули и такое...

Можно сколько угодно смеяться над нашей наивностью и недальновидностью, но когда вот сейчас, наяву, предлагается в точности та работа, о которой (простите за высокопарность) мечтал всю жизнь...

Кажется, мы и сейчас приняли бы подобные условия (шутка).

Глаза боятся, а руки делают

Не знаю, решились бы мы на этот проект, если бы сразу представляли (так, как знаем сейчас) его истинную трудоемкость. Тогда язык Си++, судя по учебным пособиям, казался нам... да, непростым для компиляции, с корявым и неоднозначным синтаксисом, сильно усложненной семантикой традиционных конструкций, но вполне сравнимым, например, с объектной версией Паскаля фирмы Borland. Так что срок, названный шефом, поначалу не вызвал у нас протеста. Однако чтение первой же действительно серьезной и подробной книги - перевода авторского определения языка [1], предложенного в качестве начальной версии для его стандартизации, повергло нас в ужас и панику. Казалось, это безумие невозможно реализовать вообще! Тогда мы поняли настоящую цену учебникам типа "Язык XXX за двадцать один день" или "YYY - это просто!". Подобные тексты (сами по себе, быть может, и неплохо написанные) оставляют за своими рамками настолько обширные области языка, избегают касаться стольких его тонкостей и особенностей, что в голове у читателя-программиста формируется зачастую усеченный и выхолощенный образ инструмента, который он собирается использовать.

Вообще, у автора вызывает некоторую настороженность, когда о сложных вещах пытаются говорить упрощенно (это касается не только программирования). Задачи, решаемые современными программными системами, очень и очень сложны. Для их создания приходится использовать адекватные инструменты, которые не могут не соответствовать сложности и ответственности задач и потому объективно не могут быть простыми. Поэтому писать о Си++ в стиле "Откройте файл `myprog1.cpp` с компакт-диска, прилагаемого к книге, и нажмите ***Ctrl-F9***. Поздравляем! Вы выполнили вашу первую программу на Си++!" - недопустимая профанация предмета.

С тех пор мы считаем, что настоящее пособие по сложному современному языку программирования общего назначения (уровня Си++ или Ada95) должно иметь форму, близкую упоминавшейся выше книге Эллис и Страуструпа,- *комментированный стандарт*. Только такая книга может дать читателю настоящее понимание языка. Да, читать и пытаться понять строгий, сложно построенный, местами даже занудный текст будет весьма непросто - но кто сказал, что профессия программиста проста? Мы обязательно сделаем такую книгу по Си++, когда его Стандарт, наконец, будет принят.

> **Комментарий 2001 года**. Стандарт принят в 1998 году - уже почти три года назад, а обещанных комментариев до сих пор нет… Собственно текст Стандарта я практически полностью перевел, надо бы засесть и за комментарии. Однако одному мне не справиться… Саша Кротов, где ты!?..

Мы подошли к делу серьезно. Три или четыре месяца мы практически не программировали. Мы изучали Эллис и Страуструпа ("Зеленую книгу") вдоль и поперек и во всех мыслимых направлениях, продумывали общую конфигурацию компилятора, выбирали построение основных структур данных и важнейших алгоритмов, предлагали и обсуждали проектные и технические решения и писали проект.

Прекрасно помню чувство гордости, которое мы испытали, увидев наглядное свидетельство наших трудов - увесистый том, привезенный Вальтером, красиво отформатированный, распечатанный на лазерном принтере (у нас их тогда и в помине не было) и даже, кажется, переплетенный. Сейчас, когда прошло уже около трех лет, очень многие наши проектные решения кажутся прямолинейными, наивными и даже неверными; некоторые пришлось менять уже в процессе реализации, но, тем не менее, проект дал необходимую основу для работы.

Этот текст, кажется, произвел достаточное впечатление на бельгийцев; они вполне убедились в уровне нашей квалификации. Тогда показалось удивительным, но некоторых простых вещей они просто не знали: например, что `typedef`-объявление не вводит новый тип, конструкции **`extern "C"`** могут быть вложенными и т.д. Не говоря уже о более специфических аспектах. Когда мы описывали в проекте технику компиляции вызовов, мы употребили термин "thunk" (короткий код для вычисления фактического параметра). Оказывается, они, сделавшие несколько коммерческих компиляторов, не знали, что это такое! С удовольствием и тайным злорадством я выписал из классической книги Гриса [2] и послал им большую цитату, объясняющую этот термин...

Первые радости

Начнем с синтаксиса (самого, казалось бы, простого аспекта, однако борьбой с ним завершаются попытки очень многих). Несколько первых впечатлений. Во-первых, язык просто очень большой. Это означает, что синтаксические таблицы - составленные вручную или построенные каким-нибудь генератором распознавателей, вроде YACC, будут довольно велики, что, естественно, замедлит скорость синтаксического разбора.

Однако при внимательном анализе оказывается, что в языке имеется сравнительно большое число «микро»-регулярностей - часто повторяющихся устойчивых последовательностей лексем. Например, пары пустых скобки: `()`, `[]`, пустой список параметров `(void)`, завершитель списка параметров `...)`

встречаются очень часто. После служебных слов **`if`**, **`switch`**, **`while`** всегда должна стоять левая круглая скобка, после **`break`** и **`continue`** - точка с запятой, а после слова **`goto`** располагаются идентификатор и точка с запятой. Таких регулярностей набирается несколько десятков, так что если рассматривать их как отдельные лексемы, объем синтаксиса заметно сокращается. Введение каждой такой "суперлексемы" экономит по крайней мере одно обращение синтаксического анализатора к таблице разбора. Усложнение распознавателя лексем (сканера), вынужденного составлять суперлексемы из пар или троек обычных лексем, при этом получается весьма незначительное; более того, если сканер во время одного вызова распознает, например, не только служебное слово **`switch`**, но и левую круглую скобку, идущую за ним, получится экономия и на числе обращений к сканеру!

Во-вторых, в синтаксисе есть неоднозначности. Это надо оценить: в Стандарте (!) языка программирования прямо написано, что некоторые конструкции можно трактовать двояко - либо как объявление, либо как оператор! В несколько упрощенном виде формулировка из стандарта выглядит так: "выражение, содержащее в качестве своего самого левого подвыражения явное преобразование типа, которое записано в функциональном стиле, может быть неотличимо от объявления, в котором первый декларатор начинается с левой круглой скобки". Классический пример: что такое `T(a);` если `T` - некоторый тип? С одной стороны, это как бы объявление переменной с именем `a`, тип которой задан как `T`. С другой - конструкцию можно трактовать как преобразование типа уже объявленной где-то ранее переменной `a` к типу `T`. Все дело в том, что в Си++ статус операторов и объявлений полностью уравнен; последние даже и называются *declaration-statements* - операторы-объявления, то есть традиционные операторы и объявления могут записываться вперемежку. Все же радости с круглыми скобками перекочевали в Си++ прямо из Си, в котором типы конструируются подобно выражениям, и тривиальное объявление можно задать либо как "`int a;`", либо как "`int(a);`". Все это понятно, но от этого не легче. И такой язык любят миллионы программистов?! Мир сошел с ума. Яду мне, яду!..

Смысл правил разрешения неоднозначностей сводится, по существу, к поразительной фразе, простодушно выведенной в "Зеленой книге": "если конструкция выглядит как объявление, то это и есть объявление. В противном случае это оператор". Иными словами, чтобы разрешить неоднозначность, следует рассмотреть всю конструкцию целиком; фрагмент "`T(a)`" для анализа недостаточен - за ним сразу может следовать либо точка с запятой, тогда выбор делается в пользу объявления, либо "что-то еще". Например, вся конструкция может выглядеть как "`T(a)->m = 7;`" или "`T(a)++;`" - это, конечно, операторы (точнее, *операторы-выражения*, в терминах стандарта). Ну а как понимать следующее: "`T(e)[5];`" или "`T(c)=7;`"? А это, будьте уверены, еще не самые разительные примеры - загляните в разд. 6.8 Стандарта.

Человеку хорошо, он ко всему привыкает, рано или поздно он разберется, но как заставить анализатор понимать эту чехарду? Пока он не доберется до точки с запятой, он, в общем случае, ничего не сможет сказать о конструкции. Друзья, не пишите объявления, которые невозможно отличить от операторов! Пожалейте компилятор, ему же тяжело! Кроме того, можно запросто ошибиться и самому...

Несколько дней прошли в бесплодных попытках выразить неоднозначности на входном языке YACC. Выход был похоже, только в организации просмотра вперед, причем на заранее не известное количество лексем. Алгоритм разбора,

заложенный в YACC, этого делать не умеет. В принципе известны и доступны системы, в которых заявлена подобная возможность, однако мы были ограничены требованием: синтаксический анализатор писать на YACCe, более того, на его версии, сделанной в одном европейском университете... Пришлось пойти на ухищрения и "сломать" классическую схему разбора: делать предварительный анализ еще на уровне разбора лексем и, встретив левую скобку после имени типа (а еще пойди распознай, что идентификатор - имя типа, а не какой-то другой сущности!), "отменять" автоматический анализ и организовывать "ручной" перебор последующих лексем, складывая их про запас в буфер.

Спасибо, в "Зеленой книге" подсказали схему такого анализа. Не знаем, как и благодарить, сами бы ни за что не придумали...

Что такое идентификатор?

Помимо неоднозначностей в синтаксисе быстро обнаружились другие неприятности. На примерах их показать сложнее, так что придется рассказывать словами.

Синтаксис языка Си++ неудобен еще и в другом отношении. Если говорить коротко, то прямое использование в формальном описании одного из базовых синтаксических понятий - идентификатора - приводит к тому, что YACC расценивает грамматику языка как некорректную и на ее основе не может построить синтаксический анализатор. Для традиционных языков синтаксическому анализатору для разбора конструкции достаточно информации о том, что в данной позиции этой конструкции может (или должен) находиться идентификатор. Более простые языки сконструированы так, что семантика идентификатора не влияет на корректность синтаксического разбора. Вид программной сущности, обозначаемой этим идентификатором (подпрограмма, переменная, имя типа, исключение, метка и т.п.), смысл данного конкретного вхождения (объявление или использование) - все это выявляется далее, как правило, являясь предметом следующей фазы компиляции - семантического анализа.

Для языка Си++ такая схема не проходит. Чтобы быть в состоянии синтаксически распознать многие конструкции, требовалась семантическая интерпретация имени. Иными словами, на вход синтаксическому анализатору следовало поставлять не абстрактную лексему "идентификатор", а результат анализа того, что именно представляет собой этот идентификатор: "имя типа", "новое имя в объявлении", "имя не-типа в выражении" и т.д. Заметим, что синтаксическому анализатору для Java - непосредственного потомка Си++ - вполне хватает понятия идентификатора без каких-либо уточнений.

Всего для Си++ получилось около десятка таких "суперлексем", а лексема "идентификатор" вообще исчезла из синтаксиса. Понятно, что лексический анализатор, который и поставляет лексемы, пришлось наделить дополнительным "интеллектом". Теперь он должен был не просто выделять из текста программы очередную лексему, но и обращаться в таблицы трансляции за информацией о том, что за идентификатор он выловил. Реально эти действия выполняет отдельный модуль, названный "расширенным лексическим анализатором". Введение дополнительного модуля не привело к усложнению компилятора в целом, так как идентификация имен так или иначе должна производиться; мы просто перенесли ее на более ранний этап компиляции. А синтаксис заметно упростился, стал более наглядным, информативным и в конечном счете более эффективным.

Компилятор как таковой: таблицы и деревья

Однако синтаксис - это мелочи жизни. Основное в любом компиляторе - это интерпретация семантики языковых конструкций, и подавляющая часть кода приходится именно на семантические алгоритмы.

Есть два основных вида семантической информации, которые компилятор извлекает из текста исходной программы. Во-первых, это информация о различных объектах, которые используются в программе (переменные, типы, функции и т.д.), причем не только об объектах как таковых, но и об областях действия, в которых эти объекты существуют (имеют смысл), а также об отношениях этих областей между собой (контекстах). Чем сложнее устроен язык, тем больше в нем правил, связанных с объектами, и тем более изощренной должна быть та структура в компиляторе, которая описанную информацию содержит. Такая структура обычно называется семантическими таблицами.

Во-вторых, компилятор должен формировать некоторый образ исходной программы - внутреннее представление программы в целом или ее некоторой части, которая в данный момент обрабатывается. Именно на основе такой структуры обычно выполняется семантический анализ программы, производятся различные оптимизации и осуществляется генерация результирующего кода. Как правило, такое внутреннее представление строится в виде дерева и потому называется деревом программы.

Эта пара - таблицы и деревья, вместе с различными алгоритмами, работающими над ними, без преувеличения составляет две трети текста компилятора. Почти вся наша работа на протяжении всех этих лет так или иначе была связана с ними.

Структура таблиц была придумана в целом по образцам из книг по теории и практике компиляции, которые в изобилии выходили у нас в 70-80-х годах и описывали, как правило, языки с относительно простой и, самое главное, регулярной структурой и несложной семантикой,- такие как Алгол-60, Паскаль, Модула-2. Многое из того, что есть в Си++, с трудом "втискивалось" в академические построения, и приходилось дополнять и развивать их. В результате таблицы представляют собой причудливую смесь классической стековой модели с дисплеем для отображения текущего контекста и наворотов вроде средств динамического перестроения контекста для обработки функций-членов классов, нетривиальной поддержки областей действия имен (namespaces), буферов для отложенной компиляции и т.д. К тому же таблицы должны быть динамически расширяемыми, чтобы быть в состоянии вобрать в себя очень большое количество имен, типичное для программ на Си++. Помучиться пришлось изрядно, и далеко не сразу таблицы заработали стабильно и надежно.

Опуская технические детали, следует сказать, что сейчас мы в целом недовольны тем, как спроектированы семантические таблицы. В свое оправдание отметим, что все "навороты" в них - вещи вполне объективные, которые так или иначе должны присутствовать в компиляторе. Наша неудовлетворенность имеет, скорее, эстетическую природу: таблицы не выглядят стройной системой, где каждый компонент точно подогнан к тому месту, которое для него предназначалось.

С деревом программы ситуация была обратной. Будучи один раз спроектированными, принципы организации дерева далее практически не изменялись. В противоположность таблицам, структура которых создавалась, чтобы непосредственно отражать контекстные отношения языка Си++, дерево

оказалось практически полностью языко-независимым. Иными словами, используя основной строительный элемент дерева - терминальный узел - можно конструировать произвольные конфигурации, отображающие конструкции любых языков программирования. Все узлы дерева имеют идентичную структуру, различаясь лишь значениями своих (немногочисленных) атрибутов. Каждый узел имеет четыре ссылки (вверх, вниз, влево и вправо), с помощью которых легко формировать "плоские" конфигурации, соответствующие тем или иным конструкциям входного языка. Как правило, горизонтальные ссылки отражают верхний уровень структуры некоторой конструкции, а вертикальные используются для поддеревьев, соответствующих элементам этой конструкции, или вложенным конструкциям.

Несомненными достоинствами такой схемы являются высокая регулярность, простота и универсальность. Дерево для любой языковой конструкции строится по единым правилам, и все разнообразие выразительных свойств Си++ приводится к строгой единообразной регулярной конфигурации, для которой очень удобно строить всевозможные рекурсивные алгоритмы анализа, трансформации и генерации.

Однако у этих достоинств есть и оборотная сторона, которую можно определить как низкий уровень структуры дерева. В чистом виде оно не несет в себе никакой семантической информации - это лишь определенная структура и ничего более. Иными словами, для дерева как такового можно определить только достаточно примитивные операции, например, "связать два узла горизонтальными ссылками", "построить из данных узлов бинарное дерево" и т.п. Любое же мало-мальски серьезное действие, учитывающее семантику того или иного поддерева, например, его перестроение в процессе оптимизации или при генерации кода, приходится программировать специально для каждого вида конфигураций. На практике это приводит к тому, что операции над деревом не располагаются в одном или нескольких модулях, а рассредоточены по всему тексту компилятора.

Этот раздел хочется завершить несколько неожиданным выводом. Наличие в компиляторе двух базовых структур - семантических таблиц и дерева программы - сейчас расценивается нами как один из самых серьезных недостатков компилятора. Эти структуры реализованы на различных принципах, работа с ними организована по-разному, однако они существуют вместе, пронизаны взаимными ссылками (можно сказать, переплетены, как корни растущих рядом деревьев) и в некоторых случаях просто дублируют друг друга. Сходная информация о структуре программы присутствует и в таблицах, и в дереве, что долго приводило к путанице, и сейчас выглядит довольно нелепо. Например, в дереве имеются узлы, соответствующие объявлениям; это естественно, так как образы объявлений могут попадать в результирующий код. Что же касается таблиц, то они как раз и составлены на основе информации, извлеченной из объявлений. Поэтому в узлах-объявлениях содержится ссылка на соответствующее слово в таблицах. Семантическое слово, в свою очередь, имеет обратную ссылку на узел "своего" объявления, которая в ряде случаев оказалась необходимой. Инициализатор переменной из объявления представляется поддеревом, на которое имеются ссылки как из узла-объявления, так и из семантического слова. И так далее... Все это работает и даже вполне эффективно, но, конечно, с точки зрения программного дизайна весьма далеко от совершенства.

Описанное построение компилятора свидетельствует о нашем честном следовании классическим образцам, а также... о нашей боязни отойти от этих

образцов. Даже подступая к языку, который заведомо отличался от "правильно" построенных, элегантных и простых языков, хорошо подходящих для книжного описания базовых концепций и методов компиляции, мы преувеличили степень универсальности решений, предлагаемых в подобных книгах.

Теперь мы это хорошо понимаем. В следующей версии компилятора все будет сделано по-другому.

Лебедь, рак и щука, или Гадкий утенок

Мой коллега и товарищ, Саша Кротов, вдвоем пару с которым мы, собственно, и сделали практически всю работу, имеет прекрасное образование (по моим наблюдениям, выпускники мехмата зачастую имеют более высокую программистскую квалификацию, чем окончившие ВМК - да простят меня мои однокашники!). Несмотря на естественное для его возраста отсутствие опыта крупных разработок, он поразительно быстро "въехал" в проект и вообще в проблемную область и очень скоро стал совершенно равноправным его участником. К этому времени он вполне осознал, насколько интереснее программировать компиляторы, чем наполнять базы данных, рисовать на экране вертящиеся фигуры или писать байты в порт и ожидать их оттуда.

Третий участник проекта был (и есть) настолько талантлив как программист, что мог с одинаковым успехом участвовать, кажется, решительно в любом программном проекте. Базы данных и сети, протоколы обменов и многозадачность, графика и издательские системы, вычислительные алгоритмы, распределенная обработка и искусственный интеллект - во всем он чувствовал себя уверенно и имел на то полное основание.

Так что шеф, глядя на нашу троицу с неподдельной гордостью, вполне мог считать, что вместе мы горы свернем. Однако далеко не все было гладко...

В компиляторе есть проектные ошибки (хотя к настоящему моменту большинство из них мы извели, поначалу их было немало). "Снаружи" эти ошибки практически никак не проявляются и не дают покоя только нам, знающим наизусть все его внутренности. Некоторые из них были просто неизбежны, так как, закладывая то или иное решение несколько лет назад, мы никак не могли представить, к чему приведет непредсказуемая эволюция языка. Другие ошибки объяснялись недостаточным опытом - практически впервые мы пытались делать то, что называется коммерческим программным продуктом и исходили только из академических представлений о том, какова должна быть архитектура компилятора.

Но самая неприятная категория проектных ошибок - это те, которые возникли из-за недостаточно тщательного анализа на начальных этапах проекта и, что хуже, из-за того, что по некоторым принципиальным вопросам имелись различные мнения. Принимать решение всегда сложно еще и потому, что чье-то мнение, как правило, приходится отвергать. Тяжело и тому, кто отвергает, и неприятно тому, чье мнение не учитывается. Зачастую бывает так, что трудно предпочесть какой-либо конкретный вариант из нескольких альтернатив просто потому, что все они достаточно обоснованы и могут быть использованы; в таких случаях необходимо чье-то волевое решение, которое все участники должны безоговорочно принять. У нас в свое время просто не хватило духу проговорить все до конца и определиться полностью по всем принципиальным вопросам. В результате некоторые существенные решения принимались "по умолчанию" тем или иным участником проекта без согласования с другими. Винить в этом,

естественно, следует прежде всего старшего участника - автора этой статьи (как самого опытного, а не самого умного!).

Так, компилятор сначала выполняет полный семантический анализ всего исходного текста, и только потом генерирует для всей программы результирующий код. Почему такая организация компилятора была выбрана третьим участником, до сих пор непонятно. Какое-то объяснение было тогда дано, но оно тут же выпало из нашей памяти, и вспомнить сейчас невозможно, а попытаться самим объяснить - не получается. Такое решение (удивительно, но принятое без всякого обсуждения) приводит к тому, что компилятор сохраняет полное дерево программы (и, следовательно, вынужден сохранять и семантические таблицы, так как они друг с другом сильно связаны) вплоть до завершения обработки всего исходного текста. Более логичным и экономичным был бы подход, согласно которому для каждой функции выполняется вся обработка, вплоть до генерации кода, после чего в структурах компилятора сохраняется только информация из ее заголовка, необходимая для компиляции вызовов. Исключение достаточно сделать для встраиваемых (inline) функций, да и то не всегда.

Сохранение полного дерева программы было необходимо, если бы компилятор "затачивался" на выполнение иных, кроме генерации кода, функций, например, на анализ программ (снятие метрических характеристик, статическое профилирование и т.п.) или в том случае, если бы мы собирались делать машинно-независимую глобальную оптимизацию на уровне входного текста. Ничего подобного в проекте не было.

Получилось так, что семантические таблицы были спроектированы, имея в виду второй, более естественный подход, а структура дерева - согласно первому подходу. Разработчику семантических таблиц (мне, попросту говоря), будучи поставленному перед фактом уже в процессе реализации, ничего не оставалось, как срочно перекроить их структуру. Крайне неприятно, но эта ситуация сохранилась и по сей день. Надо ли уточнять, что эти структуры были в свое время придуманы двумя участниками, которые в свое время не смогли (не захотели) вместе обсудить свои решения и возможные проблемы...

> **Комментарий 2001 года**. Справедливости ради следует сказать, что такое проектное решение неожиданно оказалось весьма уместным и даже естественно необходимым в «следующей жизни» нашего компилятора. Однако, в том, первоначальном, проекте оно сильно осложнило разработку компилятора, и, конечно, было недопустимым.

Подобных, более мелких, но крайне неприятных рассогласований и неувязок было много, и, самое ужасное, с течением времени их число нарастало. Возникало тяжелое ощущение того, что компилятор - это большая темная комната, а у тебя только маломощный фонарик, который в состоянии осветить небольшой аппарат - твои модули. От аппарата тянутся в темноту провода и вереницы зубчатых колес. Что делается в дальних углах, неизвестно. Иногда вокруг раздаются какие-то звуки, из темноты выступают части каких-то движущихся механизмов, назначение которых остается неведомым, даже если осветить их. Время от времени из темноты раздается голос, настоятельно требующий: "нажми на кнопку с надписью ABC", "переведи рычаг XYZ в правое положение". Что делается в комнате и как все работает вместе, понять совершенно невозможно.

Пришло время говорить о неприятном - через некоторое время от нас ушел третий участник. Он весь был ориентирован на получение результата, а не на

процесс его достижения. Само по себе это исключительно ценное качество, его наличие (подкрепленное высокой квалификацией) гарантирует успешное завершение работы в заданные сроки. Однако в данном случае оно обернулось своей худшей стороной - откровенно небрежным кодированием ("компилятор соптимизирует" - классический ответ на все замечания), принятием важных решений "на ходу", без всякого обсуждения и плохо скрываемым недовольством коллегами, которые непонятно почему копаются там, где надо скорее программировать. Главное - скорее! За один день сделать работоспособный синтаксис, за месяц - добиться трансляции программы "Hello world!". Сложность системы не играет никакой роли, все программы устроены одинаково. Модули должны взаимодействовать согласно своим интерфейсам, обсуждать и комментировать которые нет смысла, они и так сами за себя говорят - на то они и интерфейсы.

Однако компилятор - такая система, которая объективно (исключая вырожденные случаи) не может быть сделана за три месяца, даже если предположить, что найдется гений, который физически смог бы написать за этот срок нужный объем кода. Как процесс его разработки, так и процесс кодирования должен предполагать совместную работу, постоянное обсуждение всех мало-мальски существенных решений и крайне аккуратное продвижение вперед, по крайней мере, до тех пор, пока не будет достигнут этап отладки. Слишком велико число связей между всеми компонентами компилятора и невозможно предвидеть, насколько серьезными окажутся последствия самого, казалось бы, невинного решения, принятого "на проходе" как очевидное.

Скрытое напряжение в команде возрастало, и первым не выдержал тот, кто не связывал с проектом все свои помыслы. В один прекрасный день мы двое обнаружили в общем каталоге с рабочими тестами полторы сотни примеров, которые ломали компилятор, причем ломали его вроде бы на тех модулях, которые писали мы. Третий участник исчез. Мы поняли это однозначно: мое терпение кончилось, разберитесь, наконец, с тем, что у вас не работает, догоните меня, а я пока займусь другими делами.

Ошибки были исправлены примерно за неделю (половина из них оказалась "не нашими", а как раз того третьего), однако он так и не вернулся в проект никогда... Мы остались вдвоем.

Как ни покажется странным, мы с Сашей не восприняли происшедшее как катастрофу, хотя вроде бы потерю такого классного специалиста невозможно возместить. Наоборот, мы почувствовали, что у нас появилось второе дыхание, распределили "ничейные" теперь модули между собой и с подлинным энтузиазмом принялись переделывать их. К настоящему времени в них не осталось, наверное, ни единой строчки первоначального кода. Но, к сожалению, осталось несколько тех самых "волевых" проектных решений, которые были приняты без всяких обсуждений как очевидные, которые оказались впоследствии ошибочными и которые к тому времени настолько вросли в ткань компилятора, что духу и сил не хватает их из него вырезать.

Я хочу, чтобы нас правильно поняли. У нас нет к ушедшему абсолютно никаких претензий. Нас не обманули, не предали, не нарушили никаких обязательств. Более того, я вполне допускаю, что сами мы не без греха, и работа в то время шла не слишком ритмично (надеюсь, что и ему уход не принес много горечи). И если я рассказываю об этом эпизоде, то только потому, что мы сами многое при этом поняли и многому научились.

Чем меньше коллектив, тем большее, часто определяющее, значение приобретает проблема личностной совместимости - характеров, темпераментов, привычек и манер, т. е. вещей, которые прямо не относятся к профессии. Примером, близким к идеалу, можно считать Дениса Ритчи и Кена Томпсона. Вот как последний говорил об этом в выступлении при вручении ему премии имени Тьюринга: "Наше сотрудничество было образцом совершенства. За десять лет, которые мы проработали вместе, я могу вспомнить только один случай нескоординированной работы. Тогда я обнаружил, что мы оба написали одинаковую ассемблерную программу из 20 строк. Я сравнил наши тексты и был поражен, обнаружив, что они совпадают посимвольно. Результат нашей работы был намного больше, чем вклад нас обоих по отдельности". Но это, как говорится, от Бога, один случай на миллион. Каких-либо рекомендаций давать невозможно, единственное - надо быть очень и очень осторожным при формировании коллектива.

Что же касается тщательного проектирования и особенностей процесса реализации, то изначальное жесткое разбиение на модули, снабжаемые строгими спецификациями, после чего реализацию этих модулей можно отдать даже и студентам, проходит для хорошо формализуемых и не впервые решаемых типовых задач, а не для систем с предельно сложной логикой, где решительно все взаимосвязано. Традиционная этапность разработки ПО (спецификация и анализ требований, проектирование архитектуры системы, спецификация модулей, реализация и т. д.) в данном случае неизбежно размывается, модифицируется и приобретает существенно итеративный характер: проектирование (и перепроектирование) многих структур данных и алгоритмов компилятора происходит неоднократно уже на этапе реализации. Такой возвратно-поступательный процесс, как мне кажется, органически характерен для создания любой сложной программной системы, семантика которой не может быть осознана и формализована полностью на этапе проектирования в приемлемые сроки. К тому же надо иметь в виду, что в процессе работы над компилятором изменялся и *сам язык* - процесс стандартизации зачастую преподносил совершенно неожиданные сюрпризы, и многого нельзя было предугадать заранее.

Эта точка зрения, точнее, конкретный опыт, быть может, входит в противоречие с современными моделями процесса создания ПО, описанными классиками,- Г.Бучем, Э.Йоданом и другими, однако повторю еще раз, компилятор Си++ - не вполне типичная программная система, по крайней мере, с точки зрения семантической и логической сложности.

Так или иначе, мы приобрели ценнейший опыт, полностью пройдя все этапы жизненного цикла программного продукта (в том числе и его сопровождение) и набив на этом пути много шишек. Теперь, надеюсь, мы не наступим еще раз на те же самые грабли.

А вы?

Стиль программирования: на вкус и цвет товарища нет

По условиям контракта языком реализации был стандартный Си. Бельгийцы прислали свой компилятор ANSI C, но основным рабочим инструментом для нас служил gcc из системы GNU, так как он был лучше совместим с нашим любимым отладчиком gdb по формату объектных файлов. Много позже, и это ощущалось нами как внушительный успех, мы начали транслировать компилятор самим собой.

Как и полагается каждой солидной фирме, у наших партнеров имелись собственные внутренние правила и стандарты программирования. В числе необходимых для работы материалов они привезли нам документ под названием "C Coding Standards".

Трудно высказывать объективное мнение по такому тонкому вопросу, как стиль программирования. Здесь, как нигде больше, в полной мере проявляются вкусы и привычки программиста, которые очень трудно преодолеть, если они вступают в противоречие с требованиями, которым приходится следовать в работе. Зачастую расходятся мнения и участников проекта.

Я оказался в меньшей степени отравлен магнетическим воздействием системы UNIX и традициями программирования на Си, или, если угодно, находился под влиянием иной системы традиций ("правильно" построенные языки типа Алгола-68, Паскаля и Ады, большие компьютеры с "настоящими" операционными системами и т.д.), и с большим трудом привыкал к диктуемому "птичьим" языком Си стилю программирования, идущему, как мне кажется, непосредственно от личных пристрастий и привычек создателей языка. Фирменный стандарт, которому предлагалось следовать, честно воспроизводил эти "исторические" особенности, возводя их в ранг если не абсолютной истины, то безусловной нормы.

Мой коллега принадлежит к следующему поколению программистов, чье взросление пришлось на эпоху повсеместного распространения мини-машин и, стало быть, на период повального увлечения UNIXом. Поэтому он впитал дух Кернигана, Ритчи и Томпсона одновременно с базовыми концепциями вычислительной науки и гораздо раньше почувствовал себя в этой среде как рыба в воде. Понятно, что он воспринял все рекомендации и требования фирменного стандарта как нечто естественное и само собой разумеющееся.

Автор практически полностью пропустил эпоху СМ-ок, пересидев ее в машинном зале "Эльбрусов". Выдающаяся элегантность архитектуры этой системы, ее несомненная революционность в сочетании с классическими традициями программирования, положенными в ее основу, заставляли относиться к UNIX с легкой иронией - как к любопытной системе с развитым командным языком и с удачным набором небольшого числа хорошо сочетаемых базовых понятий.

А язык Си показался поначалу чуть ли не студенческой поделкой, сляпанной на скорую руку для себя и друзей, когда уже не было сил программировать на ассемблере и BCPL. Да, собственно, и сами создатели языка не слишком скрывали именно такой первоначальной ориентации Си. Своеобразное изящество, несомненный магнетизм и подлинная мощь этого языка стали осознаваться (и это очень интересно и знаменательно) только при изучении тех новых свойств, которые были внесены в него создателями Си++. В частности, знаменитая лаконичность Си - объект особенно сильной критики его противников - показала свою несомненную полезность и необходимость для механизма шаблонов. Несомненно, основанная на шаблонах парадигма обобщенного программирования А. Степанова не выглядела бы в Си++ так органично, будь этот язык столь же многословен, как Ада. (Сам Александр Степанов признавался, что его попытка создать STL для Ады провалилась прежде всего из-за чересчур «статического» характера этого языка.)

После внимательного прочтения предлагаемых соглашений о кодировании на фирму было послано длинное эмоциональное письмо с подробным анализом стандарта и выводами, смысл которых заключался в архаичности и

немотивированности большинства его требований и норм. Не буду воспроизводить все критикуемые нормы, скажу только о таком требовании, как представлять последовательности пробелов, которые используются для формирования отступов, непременно символами табуляции. Может быть, это как-то и оправдано для среды UNIX, в которой мало удобных и гибких текстовых редакторов, и символ табуляции практически всегда отображается на экране восемью пробелами, Однако на персоналках любой мало-мальски приличный текстовый редактор можно настроить на представление табуляции любым числом пробелов. Поэтому следование этому правилу на практике приводит к тому, что один и тот же текст будет выглядеть в различных средах совершенно по-разному. Невозможно перестраивать редактор под каждый файл с табуляциями.

Ответ был очень интересным. Поблагодарив, как водится, за интерес и внимание к документу, они ответили на каждый тезис моего письма. Возражений по существу не было вообще. Однако не было и согласия на отмену или изменение ни одного из критикуемых требований. Поначалу такое умолчание привело меня в бешенство. Истинный смысл такого странного ответа стал понятен намного позже.

Дело в том, что даже плохой стандарт лучше его отсутствия. Стандарт может быть устаревшим, неполным, содержать недостаточно обоснованные требования, но, тем не менее, крайне важно, чтобы все разработчики ему следовали. То обстоятельство, что вся программная продукция фирмы сделана по единым правилам, гораздо важнее в долгосрочном плане по сравнению с тем, что эти правила произвольны или несовременны. Общие правила (наряду с другими мерами) делают программу отчуждаемой от конкретного разработчика, давая возможность, скажем, сопровождать и развивать ее даже в случае отсутствия того, кто ее написал.

Поэтому допустить отступления от правил даже в одном проекте, пусть в большом и сложном (в моем письме без лишней скромности говорилось, что такой проект достоин отдельного стандарта), фирма, конечно, не могла. И проявленная ими реакция на запальчивые, при всей их обоснованности, аргументы в пользу модернизации стандарта говорит только о мудрости фирмачей, за плечами которых опыт многих проектов.

Что же касается нелепых или смешных требований, то это во многом действительно дело вкуса и привычек. Что там табуляции - в иных стандартах можно встретить и не такое! Например, в каком-то (правда, очень старом) руководстве по программированию на Фортране можно было встретить рекомендацию избегать подпрограмм, так как их вызовы приводят к большим накладным расходам. А компилятор GNAT языка Ada95, разработанный в Нью-Йоркском университете, при компиляции собственного исходного текста квалифицирует отступление от принятого стиля программирования (например, неверное число пробелов между оператором и комментарием) как… синтаксическую ошибку!

Так что и этот опыт тоже не прошел для нас даром. Следующий проект, в котором мы участвовали, начался именно со спецификации требований на стиль программирования (получился текст объемом более полусотни страниц), и особое внимание было обращено на то, чтобы все программисты ему следовали.

Программирование "наизнанку"

Решающим фактором в том, что сегодня мы имеем нечто, что можно с уверенностью назвать компилятором Си++, стало тестирование. Нам трудно судить о том, как следует тестировать, скажем, текстовый редактор или (упаси, Господи!) операционную систему, но наш опыт тестирования и отладки компилятора говорит о том, что это едва ли не самое главное во всем проекте.

Нам очень повезло. Кроме компилятора, бельгийцы заказали еще и пакет тестов на проверку компилятора (любого, не только нашего) на соответствие стандарту. Этот пакет мы и натравили на наш компилятор, точнее, на то, что мы тогда считали таковым.

Нам повезло и в том отношении, что пакет делала другая команда. Два опытных специалиста, давно занимавшиеся проблематикой тестирования именно компиляторов, на этом еще в прежние времена защитившиеся, придумали методику разработки тестов и написали формальные спецификации, а группа студентов по этим спецификациям выдавала сами тесты.

Надо понять специфику этой работы, чтобы оценить ее невероятную сложность. Есть стандарт языка (который и не стандарт вовсе, а "рабочие материалы по предварительному стандарту рабочих групп ANSI и ISO", и почти каждый квартал выходит новая версия этих "рабочих материалов"). Это кирпич в три килограмма. Каждый абзац стандарта содержит одно утверждение (а чаще несколько) относительно того или иного свойства языка. Тестовый пакет должен проверять каждое такое свойство, т. е. содержать соответствующий тест на каждое утверждение стандарта. Каждое утверждение тестируется в предположении, что другие языковые свойства компилятор реализует корректно,- протестировать все сочетания свойств физически невозможно. Предварительный стандарт, как уже говорилось, постоянно изменяется, значит каждый новый талмуд нужно просмотреть и увидеть, что изменилось по сравнению с прежним (перечень изменений рабочие группы не ведут), и внести в тесты соответствующие изменения и добавления.

Далее, текст стандарта написан, как и полагается, невероятно сложным, занудным, бюрократическим языком. По-другому нельзя, так как в стандарте требуется предельная точность, но попробуйте это перевести и понять! Примеров программ очень мало, и некоторые нужно разбирать так же, как разбирается сложная фраза на чужом языке - слово за словом. Это ни в малейшей степени не похоже на учебник по языку - никаких пояснений, никакой специальной методики изложения, никакого принципа "от простого к сложному", в любом месте текста может встретиться ссылка на термин, вводимый далеко впереди.

Наконец, предварительный стандарт - это текущий результат живых дискуссий, обсуждений, голосований рабочих групп; там сидят очень грамотные специалисты, но и они ошибаются и не все сразу видят. Поэтому в каждой версии есть (и в окончательном варианте будут) ошибки, неясности, двусмысленности, недоговоренности. Все это присутствует, наверное, в любом стандарте, но Си++ в этом отношении чемпион - слишком это сложный язык и слишком хаотически и спонтанно он проектируется. Так что, читая стандарт, следует четко осознать, почему та или иная фраза кажется тебе абракадаброй: то ли ты плохо знаешь английский, то ли недостаточно глубоко понимаешь Си++, то ли ребята из ANSI/ISO что-то напутали (а часто и то, и другое, и третье). И не с кем посоветоваться - все учебники по Си++ излагают в лучшем случае версию "Зеленой книги" 1990 г.,- и нельзя проверить свое понимание на компьютере: нет

компилятора, который реализовывал бы свойство, за которое комитет проголосовал на прошлой неделе. Еще не отлита та пуля...

В таких условиях и был написан тестовый набор, который в итоге содержал более 6500 тестов. (Можно понять, почему подобные пакеты стоят на Западе до 20 тыс. долл.!) Важно то, что до определенного момента две наши команды работали полностью независимо, никак не влияя друг на друга. В результате тесты не подгонялись под компилятор, а алгоритмы компилятора проектировались строго по семантике языка, без ориентации на то, чтобы протолкнуть его сквозь конкретные тесты. Взаимные консультации касались только обсуждения собственно текста стандарта - отправной точки для обеих групп. Только когда компилятор в основном был сделан, мы начали использовать тесты из него.

Вообще, создание тестового пакета - отдельная история, не менее интересная и драматичная, чем наша. На самом деле далеко не все было так гладко и последовательно, как описано выше. Наш шеф В.А.Сухомлин потратил очень много усилий на формирование коллектива тестовиков. Можно понять сложность задачи - непросто найти программистов, которые хотели и могли бы заниматься "программированием наизнанку" - использовать язык не для решения какой-либо задачи, а для проверки того или иного свойства самого языка! Авторы методики тестирования довольно быстро выполнили свою задачу и, не будучи знатоками Си++, отошли от проекта. Руководить той адовой работой по написанию тестов, о которой мы писали выше, пытались разные люди, но только с приходом Дениса Давыдова, аспиранта мехмата, процесс приобрел систематический и продуктивный характер. У этого одаренного и трудолюбивого парня была масса очень интересных и действительно перспективных находок и идей, связанных с тестированием ПО, и если бы не его совершенно необъяснимая и неожиданная кончина, вся эта работа сейчас наверняка выглядела бы еще сильнее и солиднее. Талантливые люди всегда уходят слишком рано...

С тех пор отладка тестов почти полностью легла на наши плечи, и мы вложили в пакет очень много своего труда. Тестовой команды как таковой уже нет - трудно рассчитывать, что студенты будут, практически ничего не получая, в течение долгого времени тянуть эту тяжелую лямку (тем более, что студенты ВМК сейчас могут с легкостью получить, пусть не слишком интересную, но гораздо менее тяжелую и неплохо оплачиваемую работу). Таким образом, мы с достаточным основанием можем говорить о том, что этот тестовый набор в значительной степени наш (мы имеем в виду только авторство - с формальной точки зрения он принадлежит заказавшей его фирме). Сейчас это прекрасный, всесторонне выверенный и протестированный набор из почти семи тысяч небольших, но строго специфицированных и единообразно составленных программ, охватывающий весь язык Си++ в соответствии с последней (декабрьской 1996 г.) версией предварительного стандарта. Не знаю, что делают с ним бельгийцы, продают ли они его и за сколько, но для нас ценность его исключительно велика, мы любим его не меньше, чем компилятор.

В какой-то статье было сказано, что нормальное соотношение разработчиков и тестировщиков на Западе - один к двум. У нас пропорция была даже выше.

Настоящая работа

Сейчас мы с ужасом думаем, что было бы, не будь у нас тестового пакета. Мы сами смогли бы написать сто, от силы двести слабо систематизированных

тестов (на большее не хватило бы времени и терпения), может быть, насобирали бы десяток-другой исходников на Си++, пропихнули бы все это через компилятор и ходили довольные и гордые тем, что наваяли. Потом программа начала бы исправно рушиться на каждой мало-мальски серьезной программе, мы в панике латали бы дыры, вскоре нам и заказчикам это надоело, и проект тихо умер бы, оставив у нас на руках никому не нужные останки того, что когда-то называлось компилятором. Судьба многих и многих проектов...

Все было по-другому. Вечером мы запускали тестовый прогон, утром (если наш жалкий SparcClassic или монструозный диск Maxtor на 300 Мбайт за ночь не дал сбой) получали протоколы тестирования, разбирали "по принадлежности" непрошедшие тесты, и начиналась настоящая программистская работа - поиск и исправление ошибок.

Как интересно проектировать структуры данных и алгоритмы! Какое увлекательное занятие - писать программы! Какое наслаждение смотреть, как они работают и как приятно видеть результаты прогонов! Это все и работой назвать язык не поворачивается - сплошные удовольствия. Программисты меня поймут. Настоящая же работа, которая требует предельных умственных усилий, от которой действительно устаешь, и которая по-настоящему вызывает удовлетворение, заключается именно в отладке. Нужно держать в голове (никакой отладчик в этом не поможет) замысловатую логику изрядного фрагмента очень сложной программы, буквально в виде движущихся образов представлять себе, как срабатывает та или иная функция для данного фактического параметра, и постоянно помнить состояние и глубину стека вызовов для кода, который кто-то тебя (или коллегу) дернул сделать рекурсивным. Кстати говоря, весь компилятор мы отладили без всяких фокусов, используя древние как мир отладочные печати (плюс десяток специально написанных функций, которые опять же печатали таблицы и деревья в наглядном виде) и примитивный по интерфейсу, но чрезвычайно удобный и мощный отладчик gdb.

Первые тестовые прогоны были кошмарны: на половине тестов компилятор выдавал вереницы жутких диагностических сообщений, которые, казалось, никогда не должны появляться, другие аварийно заканчивались знаменитой диагностикой "core dumped", те тесты, которым все-таки удалось прорваться сквозь компилятор, при исполнении выдавали неверные результаты, и лишь единицы завершались скромной фразой "test passed". Казалось, не в силах человеческих разобраться в этой каше. Однако, капля камень точит.

Поначалу-то как раз было легче - в первую очередь находились и исправлялись очевидные ляпы. Как правило, одно исправление приводило к проталкиванию десятка, а то и больше ранее неудачных тестов. Были, конечно, и "наведенные" ошибки, которые в один прекрасный день магическим образом бесследно исчезали, оставляя после себя смутное беспокойство (а вдруг, как исчезли, так и вновь появятся?). Но чем дальше двигалась отладка, тем дороже давался каждый тест. Ошибки становились все тоньше, специфичнее и тяжелее в поиске, а чтобы исправить найденную ошибку, иногда приходилось перетрясти десяток функций в разных модулях. Исправление одной ошибки не раз приводило к появлению целой серии "экранированных" ею ошибок, которые не могли проявиться до ликвидации первой ошибки. Компилятор, казалось, сопротивлялся лечению, словно строптивый ребенок.

А тут еще в самый разгар работы, когда ошибки щелкаются одна за другой, компилятор на глазах выздоравливает, словно от тяжелой болезни,- приходит новая версия стандарта. Значит, надо опять смотреть, что изменилось. Ладно

если вводится новая языковая возможность, это может быть несложно в реализации и даже приятно: когда ни у Borland, ни у gcc еще не были реализованы описатель `mutable` или булевский тип, у нас уже все работало. Хуже, если уточняются детали семантики хорошо известных конструкций, что, как правило, влечет за собой переделку базовых алгоритмов. Так, общий алгоритм сравнения типов, алгоритм обработки совместно используемых функций (одноименных функций, различающихся числом и типами параметров) и в особенности, алгоритмы, реализующие правила вызова деструкторов и обработки исключений переделывались после почти каждой новой версии предварительного стандарта. Понятно, что каждая такая переделка работающей программы вызывает поток новых ошибок, и мы откатываемся на месяц назад...

Вдобавок, изменяются сами тесты. Тестовый набор растет, охватывает все новые сферы языка, студенты становятся все искушеннее и опытнее, их тесты все изощреннее, да и мы постоянно находим в тестах ошибки, которые также становятся все тоньше и незаметнее. Не только компилятор отлаживается на тестах, но и тестовый набор отлаживается на компиляторе.

А интересно, как тестирует свои компиляторы Watcom?

Быстро сказка сказывается, да не скоро дело делается

Примерно через год после начала работы, как и следовало ожидать, мы осознали абсолютную нереальность и даже абсурдность первоначального срока. Хотя к этому времени у нас уже был сделан Проект, реализовано большинство базовых алгоритмов, программа (ее, конечно, еще нельзя было назвать компилятором) как единое целое уже начинала шевелиться, настоящее понимание языка Си++ и того, как следует делать компилятор с него, только-только появлялось. Мы поняли, что работа только начинается.

Те, кто успел поработать в советских научно-исследовательских организациях, хорошо знают цену так называемым *эскизным проектам*. При его составлении настоящего понимания того, как, собственно, следует разрабатывать и реализовывать программную систему, ни у кого нет. Основная цель заключается в том, чтобы "застолбить" работу, успокоить начальство и открыть финансирование. Если речь идет о действительно новой работе, опыта выполнения которой у авторов проекта нет, то он содержит либо общие слова и красивые схемы, либо более или менее аккуратные и тщательно продуманные предположения, опять-таки выраженные в достаточно обобщенных категориях. После сдачи эскизного проекта его, как правило, прочно забывают и разрабатывают систему так, как это подсказывают опыт и квалификация.

Наш проект, который мы через несколько месяцев представили бельгийцам, был разработан гораздо серьезнее, однако сам предмет оказался настолько сложен, что степень проработки проблем оказалась недостаточной. Время от времени приходилось возвращаться к проекту и вносить в него изменения и добавления. С одной стороны, это замедляло работу по реализации и не всем нравилось (выше я уже писал об этом). С другой - время разработки проекта даже очень сложной системы не должно быть чрезмерно большим: многого просто невозможно предвидеть - парадокс в том, что только начав реализацию, можно получить обоснованные проектные решения. К тому же программистам психологически очень тяжело недели и месяцы проводить в обсуждениях, рисовать схемы и писать тексты (между прочим, на английском языке!- в контракте специально был оговорен proper English), не составив ни строчки кода (здесь мы хорошо понимаем третьего участника). Но даже если бы подобных

задержек не было, мы физически не успели бы запрограммировать компилятор, оставаясь в пределах заявленного срока - объем программного текста, который предстояло написать, во много раз превышал то, что уже было написано. О некоторых трудностях реализации, которые ждут нас впереди, было даже страшно задумываться.

К нашему удивлению, бельгийцы легко согласились с продлением срока работы. Впрочем, они довольно плотно (хотя до времени и формально) контролировали весь процесс и могли понять, что мы даром времени не теряем и перенос сроков носит объективный характер.

Дальше начинается психология. Вообще говоря, первоначальные сроки никто никому не навязывал: предложили их мы, но они-то согласились! Поэтому, казалось бы, ответственность за тот факт, что эти сроки оказались нереальными (кажется, и они, и мы сейчас это понимали), следовало бы разделить между всеми. Однако комплекс вины чувствовали именно мы (они, как в личном общении, так и в письмах, вообще проявляли мало эмоций и крайне редко допускали неформальный стиль общения). И хотя теперь-то мы могли предложить более обоснованные и реалистичные сроки завершения проекта, этот комплекс нам, к несчастью, помешал.

К тому времени мы уже вполне убедились, что квалификация позволяет нам на достойном уровне довести проект до завершения без чьей бы то ни было помощи. Наши консультации с заказчиками сводились к вопросам о конкретном устройстве тех компонент их системы программирования, с которыми компилятор должен взаимодействовать. Даже на вопросы о формате промежуточного представления, которое, согласно фирменной документации, было их собственной разработкой, они далеко не всегда могли дать вразумительный ответ (позднее это нашло свое объяснение). Обсуждать детали языка было не с кем, собственных специалистов по Си++ у них не было, они вообще до сих пор программировали на Си самой древней версии Кернигана и Ритчи. Их помощь состояла в периодических посылках очередных версий предварительного стандарта (до тех пор, пока рабочая группа не стала выкладывать их на свой Web-сервер) и материалов очередных заседаний и дискуссий, предшествующих принятию того или иного языкового свойства. Последнее было действительно интересно и ценно, так как эти материалы, насколько мы знаем, нигде не публикуются и рассылаются только членам рабочих групп.

В общем, мы вполне могли бы чувствовать себя полноправными партнерами фирмы. Однако, несмотря на все сказанное, мы еще долгое время не могли избавиться от ощущения учеников или, по крайней мере, подмастерьев в мастерской именитого художника. И то, что мы не успевали вовремя выполнить работу, осознавалось нами как непростительная оплошность новичка, не поспевшего загрунтовать холст к приходу мэтра. К этому добавлялась не имеющая совершенно никаких оснований боязнь, что, услышав просьбы о пересмотре сроков, фирма прекратит сотрудничество с нами.

Так или иначе, мы... снова назвали срок, гораздо меньший того, какой нам действительно требовался! Сейчас мы уже не помним, скоько времени мы просили, но что срок был явно занижен, это точно.Надо ли говорить, что потом нам пришлось корректировать и этот срок, и еще один, и еще...

Когда говорят о необходимости какого-либо трудного, но необходимого решения вместо серии половинчатых мер, часто приводят поговорку: нельзя отрубать хвост собаке по частям в надежде, что это будет не так жестоко. Мы поступали в точности наоборот: вместо того чтобы с самого начала назначить

реальный срок, мы несколько раз понемногу его увеличивали. Боялись ли мы испугать фирмачей большим сроком? Наверное. Но, может быть, важнее даже то, что мы сами далеко не всегда могли в точности предсказать, сколько времени потребует тот или иной этап проекта. Нас все время подводил излишний оптимизм. Даже шеф, опыт которого в управлении программными проектами не сравним с нашим, не участвуя в реализации, не мог определить настоящих временных затрат.

Теперь, по прошествии времени, мы поняли, что получили очень важный урок. Способность определять реалистичные сроки больших программных проектов и умение их выдерживать - исключительно ценная составляющая опыта профессионального разработчика. По крайней мере, для нас этот урок не прошел даром: следующая серьезная и длительная работа, связанная с нашим компилятором Си++ (несколько слов о ней будет в заключение), была спланирована и выполнена почти идеально; сроки этапов выдерживались очень строго, и весь полуторагодовой проект завершился всего лишь с одно-полуторамесячным отставанием относительно первоначальных сроков. И это несмотря на то, что система состояла из нескольких компонент, в проекте участвовало около десятка человек, большинство которых впервые имело дело с разработкой таких системных программ, как ассемблеры, редакторы связей, эмуляторы и т.п., а аппаратные спецификации изменялись в течение всей работы.

Поистине учиться можно только на собственных ошибках!

Кризис

Работа вошла в режим хронического отставания. Примерно в середине описываемого периода сложилась особенно тяжелая ситуация. Процесс реализации, т. е. непосредственного программирования, постепенно набирал обороты, потоком пошли ошибки, которые следовало скрупулезно выявлять и ликвидировать. Это означало многие часы монотонной и тяжелой работы в отладчике. Однако многие ошибки показывали просчеты в наших проектных решениях; значит, нужно было откладывать в сторону отладку и опять проводить дни в обсуждениях. Наша работа приобрела характер прыжков из стороны в сторону, мы в буквальном смысле не знали, за что хвататься.

Процент пройденных тестов рос очень медленно, появлялись все новые и новые неожиданные ошибки. Мы же, несмотря на увеличивающееся отставание, уже не хотели заниматься поспешным "латанием дыр". Если ошибка свидетельствовала о некотором недочете проекта, мы, не задумываясь, пересматривали соответствующее решение и переписывали все места в компиляторе, относящиеся к нему. Образно говоря, если и приходилось делать в компиляторе "заплату", то мы стремились наложить ее прочно, захватывая стежками большие области здоровой ткани.

Проект, кряхтя и обливаясь потом, переваливал экватор. Сейчас все происшедшее представляется едва ли не закономерным, но тогда при взгляде со стороны могло выглядеть почти как катастрофа. Руководство фирмы, естественно, не вникавшее в подробности процесса и ориентирующееся только на формальные аспекты - постоянно переносимые сроки завершения очередных этапов,- начало проявлять недовольство.

Своеобразным буфером для нас в этих условиях стал Вальтер. Несмотря на свое сравнительно высокое положение в компании, большую занятость (мы видели это по его постоянным командировкам по всему земному шару - от

Америки до Австралии) и широкий диапазон сфер деятельности, он довольно глубоко вникал в наши исходные тексты, проводил дополнительное тестирование, анализировал некоторые проектные решения и давал рекомендации. Хорошо зная текущее состояние проекта, он, видимо, пытался сгладить негативное впечатление от постоянного переноса сроков. Давно перестав программировать, он не потерял квалификации как разработчик и в некотором смысле показывал пример того, как можно сочетать в одном лице менеджера европейского уровня и программиста, не гнушающегося рутинного процесса тестирования чужих программных текстов.

Тем не менее, и в его отношении к проекту стало проявляться недовольство. Первые же признаки этого повергли нас в панику: неужели мы переоценили свои возможности, и нам не по силам этот проект? Однако ничего сделать мы уже не могли: работа уже полностью захватила нас, процесс постепенно приобретал собственную внутреннюю логику, набирал инерцию и как бы сам тянул нас вперед. Наши эмоции, чье бы то ни было недовольство и вообще все сопутствующие обстоятельства все меньше и меньше влияли на ход работы. Однако отсрочка с закрытием каждого очередного этапа означала соответствующую задержку выплат.

Примерно через полгода самый трудный этап был преодолен и компилятор почти целиком вышел на стадию тестирования и отладки. Теперь по проценту пройденных тестов и по скорости прироста этого процента можно было более или менее достоверно судить о перспективах завершения всего проекта.

Вдруг начались задержки с платежами. Очередной этап пусть с опозданием, но завершается, программные тексты переданы, документация представлена,- словом, формальности соблюдены, однако под различными предлогами нам предлагается подождать. Постепенно эта практика приняла постоянный характер. Или они решили нас проучить? Как бы то ни было, мы были в бешенстве. Не в состоянии выплатить даже такие откровенно нищенские суммы - как это можно объяснить?

В один из последних приездов Вальтера мы решили продемонстрировать твердость. Никаких ультиматумов ставить, конечно, не собирались, однако хотели дать понять, что нам не нравится характер отношений с фирмой. В переписке, предшествующий его визиту, мы настоятельно просили увеличить оплату, однако получали весьма уклончивые ответы. В первую же встречу мы заявили, что суммы, которые мы получаем, существенно ниже типичной по Москве зарплаты среднего программиста (это было действительно так), а мы к тому же не средние программисты. Кроме того, получаем мы их последнее время крайне нерегулярно. Поэтому, чтобы обеспечить хотя бы сносное существование, мы вынуждены искать дополнительную работу. По этой причине мы теперь не сможем уделять проекту все свое рабочее время. (Тем самым мы заранее предупреждали о возможности очередной задержки.)

Последняя часть тирады была блефом. Мы не могли себе представить, что в Москве найдется для нас работа, связанная с созданием компиляторов, а делать что-то еще крайне не хотелось (см. начало статьи). На самом деле компилятор занимал все наши мысли, только с ним мы связывали все наши дальнейшие перспективы, так что переключиться на что-то иное было почти невозможно. Понятия же "рабочего времени" для нас давно не существовало - рабочим было почти все время, свободное от сна и от принятия пищи.

Никакого содержательного ответа мы не получили. Вальтер выслушал нас молча, лишь иногда понимающе кивал. Потом спросил: "Какова примерно

зарплата программиста в Москве?" Мы назвали сумму. Он последний раз кивнул и замолк. Разговор закончился.

Следующие дни мы обсуждали технические аспекты проекта. Казалось, он был доволен состоянием компилятора и тем, что в конце тоннеля, наконец, стал виден свет. К больному вопросу мы не возвращались. Его отношение к нашему демаршу так и осталась загадкой. То ли это не входило в его компетенцию, и давать какие-либо обещания он просто не имел права? Или что-то мы должны были понять сами, без каких-либо объяснений? Кое-что прояснилось позднее, но тогда было от чего впасть в недоумение. Как бы то ни было, мы попали в дурацкое положение. Получилось, будто между нами произошел следующий диалог:

- Будете платить? - Нет. - Ну ладно, тогда мы будем работать бесплатно.

Как отремонтировать подгнивший дом

Вскоре после того, как компилятор "задышал" и приобрел относительную стабильность, мы стали систематически проводить его профилирование. GNU'шная программа gprof выдавала длинные таблицы временных затрат отдельных функций, по этим таблицам мы рисовали огромные, на несколько листов, графы реальных взаимосвязей модулей, пытаясь найти резервы быстродействия. Первый же анализ показал, что около 40% времени тратится на операции со строками и библиотечные функции ввода-вывода. Сначала это показалось естественным - любая идентификация именованного объекта в программе предполагает сравнение имен. Поиск имени в таблицах - одна из базовых операций в любом компиляторе, и даже используя технику хеширования, избежать прямого литерального сравнения идентификаторов невозможно. Однако цифра показалась нам все же чрезмерно большой. Исправить положение без разрушения компилятора было крайне сложно, так как всевозможные операции с именами буквально пронизывали все модули. Это не являлось проектной ошибкой - полностью локализовать работу с именами невозможно, так как сама семантика языка определяет необходимость оперировать с именами практически на всех стадиях компиляции.

Известно, что у деревянного бревенчатого дома обычно первыми подгнивают нижние венцы, имеющие постоянный контакт с почвенными водами. Чтобы отремонтировать подгнивший дом, вовсе не обязательно раскатывать его по бревнышку. Делают так: определяют самый нижний здоровый венец, подводят под него домкраты (под каждый из четырех углов) и немного приподнимают ими весь дом. После этого заменяют отслужившие бревна новыми и опускают на них верхнюю часть.

Точно по такой же схеме обошелся с компилятором мой коллега. Он "вынул" из него старую схему хранения имен и заменил ее на усложненную, но более эффективную. Ключевой момент новой схемы состоял в обеспечении присутствия каждого имени в семантических таблицах в единственном экземпляре. Тогда вместо сравнения литеральных изображений имен достаточно было сравнивать указатели на эти представления. Алгоритмы модулей, использующих операции с именами, никак не изменились, однако в некоторых местах пришлось заменить тип `IDENT`, представляющий "старый" идентификатор в таблицах, заменить на `xIDENT` - прямой указатель на единственную копию данного имени. Эту операцию можно было бы сделать одной командой контекстной замены, но никакой редактор не смог бы разобраться, где именно следовало ее производить, а где - оставить по-старому... В очередной раз мы

"руками" перещупали весь компилятор. После четырех дней непрерывного труда компилятор разогнался на 25% (наглядная стоимость литеральных сравнений строк в большой программе)!

Фрагмент модуля с усовершенствованной версией обработки имен с тех пор украшает комментарий:

```
/* Krotoff is a _very_ clever guy! */
```

Товарища не похвалишь, так и он тебя не похвалит.

Последнее прости

Последние месяцы мы работали, не получая от фирмы вообще ничего, кроме писем. Постепенно ситуация с отсрочками платежей начала проясняться. У фирмы возникли финансовые затруднения. В подробности нас не посвящали, но они были, похоже, достаточно серьезными. Компания применила радикальный, но, видимо, стандартный метод выживания в подобных условиях - разделение, при котором убыточные подразделения выделяются в самостоятельные (дочерние) фирмы.

Однако для нас ничего не изменилось, и, несмотря на неопределенность, работу мы не бросали. По-прежнему примерно раз в две недели, а то и чаще мы отсылали текущие версии компилятора и тестов (snapshot - "моментальный снимок" нашей работы), Вальтер делал собственный прогон компилятора, присылал свои вопросы и замечания. Последний этап работы прошел без особых событий, в тяжелой и монотонной работе.

В ноябре 1996 г. мы получили так называемый "Milestone Certificate" - официальное подтверждение о завершении и принятии очередного этапа работы и вместе с ним - всего проекта в целом. Все взаимные обязательства были выполнены, фирма не имела к нам никаких претензий (еще бы имела!). Finita...

Компилятор готов, более чем 3-летний марафон успешно завершен! Однако это эпохальное событие прошло для нас незамеченным. Мы просто не в силах были остановиться - компилятор еще давал ошибки на семи процентах тестов, последние версии стандарта не были просмотрены, и вообще еще много чего нужно было доделать... Все как обычно.

В конце года ситуация, казалось, начала поворачиваться к лучшему. Фирма не умерла, наоборот, они решили создать филиал в другой стране, который занимался бы именно компиляторами, в том числе и нашим. Планировалось отдать компилятор на бета-тестирование в один европейский университет, готовый попробовать его в работе. Надо было готовиться к сопровождению этого процесса. Кроме того, нам предлагалось прямо с января начать новый проект - перепроектировать компилятор, чтобы он генерировал промежуточное представление нового формата, который фирма уже давно разрабатывала и даже продвигала как некоторый стандарт.

Все это выглядело очень перспективно и интересно. Мы были уверены в своих силах и собирались не повторить ни одной из прежних ошибок. Теперь мы могли вполне обоснованно определить трудозатраты и, конечно, потребовали бы более адекватное вознаграждение. Мы ждали известий о филиале и документацию по новому формату.

Внезапно что-то застопорилось. Письма стали короче и неопределеннее, ответы на наши вопросы - уклончивыми: мол, создание филиала непростая работа, документы пришлем после подписки о неразглашении. (А как давать

подписку, когда не заключен контракт и неизвестно, что, собственно, придется делать и в какие сроки?) Постепенно переписка заглохла.

Тягостная пауза продолжалась несколько месяцев. Надо было что-то решать, но в создавшихся условиях практически любое решение означало окончательный разрыв с фирмой. А мы... да, боялись этого: нам казалось, что как разработчики компиляторов мы больше никому не интересны.

Несмотря на уже неприличное молчание Вальтера и полную неопределенность с будущим проектом, мощная инерция огромного программного текста, который, подобно тяжелому составу, даже после экстренного торможения проходящему юзом до километра пути, тянула нас вперед. Все новые и новые доработки, исправления, 93% успешных тестов, 95, 97... Компилятор, хотя и был официально сдан, все улучшался и улучшался. В результате мы довольно существенно продвинули и тесты, которые в целом, как нам казалось, уже вполне можно было считать программным продуктом.

У нас возникло ощущение, что, может быть, следует переправить бельгийцам то, что мы сделали за время после формальной сдачи. Наша (конечно же, несколько наивная) логика была такова: последняя принятая ими версия тестового пакета содержит ошибки. Теперь мы их исправили. Если мы оставим исправления при себе, то, продавая пакет с ошибками, фирма подмочит свою репутацию, что косвенно ударит и по нашему реноме. Мы несколько раз спорили друг с другом; наконец, решили не посылать сразу исправления, а сначала написать письмо с предложением: не хотите ли взять у нас новые версии компилятора и тестов? Тем самым, быть может, мы вынудим их раскрыть свои планы.

Ничего подобного не случилось. Ответ Вальтера был по-своему знаменателен. Он написал: "мы всегда готовы получить от вас новые версии".

Больше никаких контактов между нами не было. Все было кончено.

Любимое дитя

На душе было тяжело. Компилятор сдан, но его дальнейшая судьба абсолютно неизвестна. Если бельгийцы намереваются пустить его в дело, то первое, что они должны сделать, как и обещали,- отдать на бета-тестирование. Полное молчание. Мы нисколько не боялись стороннего тестирования (уж сколько мы сами трепали его на всевозможных тестах!), наоборот, были бы несказанно рады, что у компилятора появляются какие-то перспективы. Но тогда, как бы ни был компилятор хорош, у пользователей обязательно должны были появляться проблемы, вопросы о непонятных ошибках и т.д... Никаких известий.

Словно послали учиться за границу единственного ребенка, а от него ни ответа ни привета.

Может быть, мы им так надоели, что они решили дальше работать с компилятором сами? Сомнительно. Несмотря на то что Вальтер в свое время продемонстрировал нам высокий уровень анализа нашего программного текста (даже ошибки у нас находил!), вряд ли, учитывая их непростое положение, они сейчас способны сами вести проект. Нет у них своих специалистов по Си++, а сформировать для поддержки бета-тестирования новую команду они просто не в состоянии. В любом случае, если бы они приняли определенное решение, рано или поздно они должны были объявить об этом публично. Однако их Web-страница наводила уныние, не меняясь уже больше года. Все это время на ней красовалось сообщение: "В конце года (какого? - авт.) у нас будет компилятор

Си++"... Она и сейчас, когда прошло еще несколько месяцев, не изменилась ни в одном символе.

Единственное, что можно было еще предположить, - они продали исходный текст какой-нибудь третьей фирме. Тогда, конечно, мы уже никогда не узнаем, по чьим рукам пошел наш компилятор... Но и в это не очень верилось. Скорее всего, сейчас им просто не до компилятора.

Как бы вы поступили в подобном случае?

Долгое время мы просто переживали и... продолжали работать над компилятором. Как обычно, именно после официальной сдачи проекта обнаруживаются и исправляются ошибки из серии "непонятно, как раньше программа вообще работала", возникают новые плодотворные идеи и в течение нескольких дней воплощаются в программный текст. Появилась еще одна редакция предварительного стандарта, и те изменения, которые мы в ней увидели, были немедленно отражены в компиляторе. И в один прекрасный момент мы вдруг осознали, что:

- теперь компилятор соответствует самой последней редакции предварительного стандарта, а не версии годичной давности, как было зафиксировано в контракте;
- мы исправили несколько ошибок, выявленных уже после сдачи, и некоторые из этих ошибок были довольно серьезны;
- некоторые базовые алгоритмы были заметно улучшены в плане эффективности; компилятор заработал быстрее.

Таким образом, мы уже существенно ушли от той версии, которую сдали бельгийцам, и почувствовали, что находимся на правильном пути. Последним толчком для нас послужило предложение от одной московской фирмы включить компилятор в задумываемую ими систему программирования.

Но ведь мы не можем распространять от себя то, что формально нам не принадлежит! Следовательно, *нужно сделать компилятор нашим*. Из того, что рассказано выше, можно понять, что у нас давно чесались руки многое переделать. Теперь для этого возникли все условия. Часть пути уже пройдена, следует определить последующие шаги в этом направлении. Несколько дней мы провели в подробных обсуждениях. Мы не специалисты в юридических аспектах, относящихся к проблемам собственности, но нам было понятно, что изменения в компиляторе должны, с одной стороны, затрагивать если не все, то большинство его основных алгоритмов, и с другой - "внешний вид" программного текста также должен быть сильно модифицирован. Функциональность программы, естественно, останется (все компиляторы, по большому счету, делают одно и то же), но ее внутренности должны в значительной степени измениться. Это должен был быть *новый* компилятор.

Читатель простит некоторую сдержанность при описании этих аспектов работы. Скажу только, что в новой версии мы умудрились изменить даже стиль программирования, не говоря уже о более простых вещах. Что же касается переработки алгоритмов, то, помимо уже описанных выше изменений, мы оптимизировали формальное описание синтаксиса входного языка, сделав разбор выражений примерно на 20-25% быстрее, доработали механизм шаблонов, включив в него массу нововведений, появившихся за последнее время, реализовали новую схему работы с именами (см. главку "Как отремонтировать подгнивший дом") и усовершенствовали как компиляцию исключений, так и

алгоритмы времени выполнения, связанные с обработкой исключительных ситуаций. Сделано еще много заметных изменений, и эта работа продолжается.

Мы перенесли компилятор на персоналки и заставили его работать в среде Windows'95 (правда, без формирования объектного кода - генератора для платформы Intel у нас пока нет).

Наконец, мы решили проблему с форматом промежуточного представления, которое порождает компилятор. Это очень интересная история. Читайте дальше.

Confidential

Наша система - не традиционный компилятор, порождающий объектный код, а так называемый **компилятор переднего плана** (front-end compiler), который в качестве результата своей работы формирует образ исходной программы на некотором промежуточном языке. Далее этот образ обрабатывается отдельной компонентой - генератором кода (back-end). Это обычная схема, давно принятая в многоязыковых системах программирования. Так как промежуточное представление выбирается единым для всех входных языков, то в системе достаточно единственного генератора кода, что исключает затраты на реализацию генератора для каждого отдельного компилятора. Кроме того, можно разработать несколько генераторов кода с единого внутреннего представления для различных аппаратных платформ, получив тем самым многоплатформную систему программирования. По этой схеме организована система gcc, похожим образом устроены и продукты семейства TopSpeed и десятки других.

Промежуточное представление, которое использовали бельгийцы в своих компиляторах (это, по существу, специальный язык, который можно назвать обобщенным ассемблером), было разработано довольно давно, выглядело несколько архаично, но для него было сделано несколько работающих генераторов для платформ Intel, Motorola, Sparc и менее известных процессоров. Спарковский генератор они и передали нам для использования совместно с создаваемым компилятором, специально оговорив недопустимость его копирования. На документации по промежуточному языку красовались жирные штампы "Confidential". Это вызывало уважение и некоторый трепет. Перед нами как бы приоткрыли дверь в святая святых компании - поделились своим ноу-хау.

Когда произошло все то, о чем было написано выше, и мы начали интенсивно переделывать и дорабатывать компилятор, стремясь сделать его полностью "нашим", перед нами, словно чугунный рельс,- ни обойти, ни сдвинуть - все время стояло это безальтернативное, как хлопок двери, слово,- "Confidential". В самом деле, пусть мы переписали компилятор, пусть его исходный текст сильно изменился, но он, тем не менее, порождает код, формат которого является чужой собственностью,- как мы можем считать такой компилятор своим? Придумать собственное промежуточное представление или адаптировать, например, внутренний код gcc - он, как и весь проект GNU, имеет статус freeware - конечно, можно, но сколько времени это займет? А соответствующая переделка компилятора сравнима с созданием нового.

Проблема встала особенно остро, когда нам предложили включить переработанную версию компилятора в состав системного ПО для нового микропроцессора. Особенно переживал мой коллега, не успевший набраться советского правового нигилизма. Однако именно он и нашел выход. Точнее, не

выход, а разгадку, поскольку, как оказалось, проблемы в действительности не было.

Был краткий период моральной усталости от отладочной гонки, которая выглядела бесконечной (последние пять процентов ошибочных тестов поддавались с невероятным трудом и требовали все новых правок). Мы задумались о будущем и начали прикидывать, как могла бы выглядеть совсем новая версия компилятора. Мы начали интенсивно искать в Интернете все, что так или иначе касалось компиляции, генерации кода и языка Си++. Как ни странно, больше всего информации оказалось о методах генерации. И вот в один прекрасный день Саша натолкнулся на работу Джонсона [3] о реализации одного из первых компиляторов Си - проекте Portable C, относящегося к концу 70-х годов. Это была статья в каком-то древнем формате с подробным описанием проектных решений и описывающая, в частности, подход к генерации кода. Мы не глядя распечатали ее и ахнули: в ней были расписаны основные коды бельгийского внутреннего представления, который мы помнили наизусть! Два дня ушло на лихорадочный поиск и запросы во все стороны, где можно найти исходники Portable C. Нашлись, родимые, рядышком, у какого-то коллекционера в Финляндии! И что же? Похожие названия команд, те же кодировки и почти те же самые заголовочные файлы, что и у бельгийцев!

Теперь мы поняли причины неуверенности в ответах на вопросы об особенностях промежуточного представления - это был *не их* формат. Многие детали так и остались тогда непроясненными. В начале работы нам приходилось познавать промежуточное представление, по существу, полностью самостоятельно, если угодно, используя "проекционный подход" В.Ш.Кауфмана: мы написали больше сотни тестов на Си, пропускали их через фирменный компилятор Си и изучали порождаемый промежуточный код, сравнивая "проекцию" с оригиналом - исходным текстом.

Не будем гадать о том, почему фирма взяла за основу своего промежуточного представления формат Джонсона. Для своего времени это было естественное и, наверное, правильное решение, и, конечно, их нельзя упрекнуть в некорректности - статья известна всем, она до сих пор входит в комплект документации по "Seventh Edition release of the UNIX operating system" компании Bell Telephone Laboratories, а исходные тексты Portable C общедоступны.

Однако для нас ситуация изменилась радикально. Кто запрещает нам проделать, по существу, то же самое? Теперь, совсем немного переработав генерирующую часть компилятора (стоит ли говорить, что за неделю это было сделано), мы можем (можем?) честно и открыто, в противоположность бельгийцам, говорить, что наш компилятор порождает промежуточное представление, формат которого соответствует формату, используемому в таком-то компиляторе (исходные тексты которого общедоступны) и описанному в таком-то году в такой-то известной статье такого-то известного автора. Это вполне соответствует общепринятой практике. Если у нас будет (а у нас будет) собственный генератор, по этому промежуточному представлению порождающий код для некоторой программно-аппаратной платформы, то мы с полным основанием можем считать наш компилятор нашим.

Это сладкое слово - свобода!

Надо бы зарегистрировать его в РАПО...

Заключение. Полетит?

К настоящему времени (конец 1997 года) мы далеко ушли от версии, сданной бельгийцам в конце прошлого года. Теперь компилятор соответствует последней, декабрьской версии Предварительного Стандарта и успешно проходит примерно 98% всех тестов. Заметно быстрее работает синтаксический разбор, почти полностью реализованы шаблоны. Наконец, теперь он перенесен на платформу Intel (в виде консольного приложения для Windows'95), а в конце года должен заработать наш собственный генератор кода для Win32.

Мы подготовили эскизный проект совершенно новой версии компилятора, надеемся, свободного от проектных ошибок, которые уже невозможно выковырять из теперешней версии и в котором заложены очень заманчивые решения (и, конечно, новые ошибки!).

Наконец мы получили опыт, ценность которого безмерна. Такими малыми силами полностью пройти проект почти предельной сложности (теперь мы можем утверждать это совершенно определенно), получить в итоге работающую программу с приличными характеристиками - это было огромным уроком, некоторые фрагменты которого я и постарался показать. Кстати, задержка реализации компилятора (значительная по сравнению с первоначальными сроками, но вполне естественная, если принять в расчет его действительную трудоемкость) дала один положительный эффект: мы смогли на практике оценить и оперативно пересмотреть многие проектные решения, опробовать несколько вариантов и выбрать наилучший в конкретных условиях.

Знакомые ребята из одной московской фирмы, разрабатывающей программно-аппаратные комплексы на основе микроконтроллеров, некоторое время назад написали компилятор ANSI Си для одного семейства однокристальных микроконтроллеров. Их история оказалась несколько похожей на нашу, что может говорить о типичности явления. Они делали компилятор по заказу известной американской фирмы, в контракте с которой был пункт об оплате всей работы после проведения тестирования. Сроки тестирования никак не оговаривались, а проводить его должны были сами заказчики. Через год компилятор был полностью готов (вместе с библиотеками, отладчиком, макроассемблером, программным эмулятором процессора и средой разработки!), однако, к этому времени ситуация изменилась: американцы, видимо, потеряли интерес к разработке и... просто не стали проводить тестирование! Придраться было не к чему, буква контракта нарушена не была. В результате фирма осталась без денег, правда, с компилятором, который теперь стал, естественно, их собственностью.

Но... нет худа без добра. Теперь они распространяют его сами и продают до шести комплектов в месяц. У нас, в России, продается компилятор, разработанный в России же, и покупают его наши пользователи! Как хотите, а это здорово.

Однако их опыт не слишком подходит нам. Ниша, образуемая системным программным обеспечением для микроконтроллеров, в гораздо меньшей степени насыщена такими инструментами, как компиляторы языков высокого уровня. Поэтому у ребят остаются неплохие шансы выйти и на мировой рынок, который, конечно, существенно шире. Что же касается инструментов общего назначения, то здесь конкуренция очень жесткая. Примеры превосходных систем программирования на Си++ для любых платформ известны всем. Что из того, что наш компилятор лучше соответствует стандарту? Зато он не интегрирован с редактором, с отладчиком, у него нет "Менеджера Проектов" и других полезных

штучек. Даже несмотря на наличие собственного генератора кода он, по существу, остается полуфабрикатом, пригодным для включения в какую-нибудь интегрированную среду, в которой есть все, что надо, но по недосмотру не оказалось компилятора. Сами мы такую среду не напишем - мало сил, не та квалификация, и вряд ли у нас это получится профессионально (каждый должен заниматься своим делом). Кому нужен такой продукт?

Правда, не все так уж плохо. Весь последний год мы работали по контракту с небольшой, но серьезной российской фирмой, создающей заказные программно-аппаратные комплексы и спроектировавшей собственный (sic!) специализированный процессор (опытные экземпляры вот-вот появятся; еще один пример того, что не все занимаются проталкиванием импортных решений!). Мы принимали участие в разработке системного ПО для этого процессора и адаптировали для него свой компилятор. Что получится из этого проекта, пока не известно. Многое зависит от того, как код, сгенерированный нашим компилятором, заработает на реальном "камне", а не на эмуляторе, и от того, сможет ли фирма заинтересовать этой разработкой потенциальных заказчиков. Однако факт остается фактом: наш компилятор получил (пусть даже очень небольшое) признание, оказался востребован.

Однако принципиально ситуация не изменилась. Возможное использование компилятора в одном проекте, к тому же еще не доведенном до конца, никак нельзя назвать успехом. Вопрос "летает - не летает?" по-прежнему остается без ответа и по-прежнему мучает нас.

Тем не менее мы остаемся оптимистами. Самолет уже поднят из подпалубного ангара и выведен на стартовую позицию, крылья разложены, двигатели работают, прогреваясь, аппарат наполняется мелкой дрожью, которая передается пилотам, уже включившим все приборы. С вышки вот-вот прозвучит команда, выводящий в наушниках махнет флажком и отбежит в сторону, двигатели взревут, переходя в режим форсажа, и истребитель, подброшенный дугообразным завершением палубы авианесущего крейсера, почти вертикально уйдет в небо.

Постскриптум

Вот совпадение! Именно в те дни, когда я правил окончательную версию этого текста и, в частности, горевал о тяжелой судьбе нашего бедного компилятора, сгинувшего в неизвестность, мы (окольными путями, практически случайно) получили два свежих рекламных листка, выпущенных нашими бельгийскими заказчиками. В типичной для таких текстов разудалой манере, с обилием восторженных эпитетов и восклицательных знаков в них представлялись... компилятор Си++ (в составе многоязыковой системы программирования) и уникальный тестовый пакет, "вобравший в себя двадцатилетний опыт фирмы"...

Я почувствовал, что моя спина как бы сама собой стала несколько прямее...

Литература

1. Грис Д. Проектирование компиляторов для цифровых вычислительных машин: - Пер. с англ.- М.: Мир, 1969.

2. C.S.Johnson. A Tour Through the Portable C Compiler. http://plan9.bell-labs.com/7thEdMan/vol2/porttour.bun.

3. Эллис М., Строуструп Б. Справочное руководство по языку программирования C++ с комментариями: Пер. с англ.- М.: Мир, 1992 - 445 с., илл. ISBN 5-03-002868-4.