

Вадим Исаев

Краткая история языка Паскаль и его создателей



Препринт — www.freepascal.ru

2020

Оглавление

0. Введение.....	3
1. История до истории. Откуда возник Паскаль.....	4
2. А вот теперь сама история. Вирт и его Паскаль.....	14
Наконец-то про Паскаль.....	19
Критика языка Паскаль.....	24
Стандарты Паскаля.....	25
Этапы трудовой биографии Никлауса Вирта.....	27
3. История после истории. Наследники Паскаля.....	32
Ответвление — UCSD Pascal.....	32
Ответвление — «Адское» программирование американской военщины ☺.....	34
1-ое поколение наследников — Apple Pascal, Borland TurboPascal и MS QuickPascal.....	43
2-ое поколение — Delphi (ObjectPascal).....	53
3-е поколение — VirtualPascal, TMT Pascal, GNU Pascal, FreePascal.....	58
Ответвление — PascalABC.NET.....	68
Дети Паскаля — MODULA.....	71
Внуки Паскаля — Oberon, Component Pascal, BlackBox и прочее, прочее, прочее.....	87
Дистрибутивы Оберона.....	95
В общем и целом про Оберон.....	112
4. Иерархия языков.....	116
5. Сравнение производительности.....	117
6. Ссылки и книги.....	119
Приложение. Список программ входящих в архив с примерами....	122

0. Введение

Сразу предупреждаю — не стоит ссылаться на эту книгу, как источник строгих и неоспоримых исторических фактов ☺, поскольку здесь представлены и не всегда серьёзно интерпретированы не только факты, но так же многочисленные слухи и сплетни, которые обычно сопутствуют, а часто и превышают по объёму, любое интересное дело или выдающегося человека. Информация о жизни Никлауса Вирта получена из его интервью [1], данном в марте 2018 года в Цюрихе Елене Тришиной.

Эта книга написана для тех, кто сталкивался в своей жизни с языком программирования Pascal (Паскаль) в том или ином виде. Это не учебник. По крайней мере не учебник по языку Паскаль, потому что учебников на сегодняшний день написано столько, что ими можно пару недель топить печь в сельской местности. ☺ Здесь описана история жизни языка и почти всех его ответвлений в ту или иную сторону, начиная от легендарных времён и вплоть до наших дней (для тех кому покажется, будто я соврамши по поводу наших дней — см. год издания книги ☺). Так сказать, для общего развития, чтобы программирование вам не казалось скучным делом.

Кроме Паскаля здесь описываются языки, которые тем или иным образом с ним связаны — Ada, Modula, Oberon и т. п. Для них дано краткое описание работы, а так же некоторое количество программных примеров. Если вы ими заинтересуетесь, то можете спокойно читать учебники по этим языкам, т. к. их основы уже будете знать по книге.

Итак, «Peignez généreusement et sans hésitations pour garder la fraîcheur de la première impression. Ne vous laissez pas intimider par la nature, au risque d'être déçu du résultat.»¹ ☺

Camille Pissarro



¹ Без раздумываний рисуйте, щедро малюя красками свежесть первого впечатления. Не страшитесь, по своей простоте, быть разочарованными конечным результатом.

Жакоб Абраам Камиль Писсарро

1. История до истории. Откуда возник Паскаль

Итак, «давным давно, в одной далёкой-далёкой галактике...». Действительно, о языке Паскаль речь нужно вести очень издалека и даже не с самого Паскаля. Кто-нибудь из вас задумывался — а почему сегодня в мире существует столько языков программирования? Согласно [2], их количество от 2 500 до 10 000, т.е. никто толком даже подсчитать не может. И если ограничится только семействами языков — всё равно будет много. Клянусь своей треуголкой, для этого есть очень веские причины. Ведь не просто так человек проснулся утром, выпил кофию и подумал «а не создать ли мне новый язык программирования, а то что-то жить скучно стало...».

На заре компьютерной эры труд программиста, без всякого преувеличения, был очень тяжёл. Вот, к примеру, ниже представлен первый американский компьютер «ЭНИАК». Именно американский, а не первый в мире, как любят триндеть сами американцы. Упрекать их за это не стоит, всем известно: не соврёшь — рассказ не получится, спросите у любого рыбака...

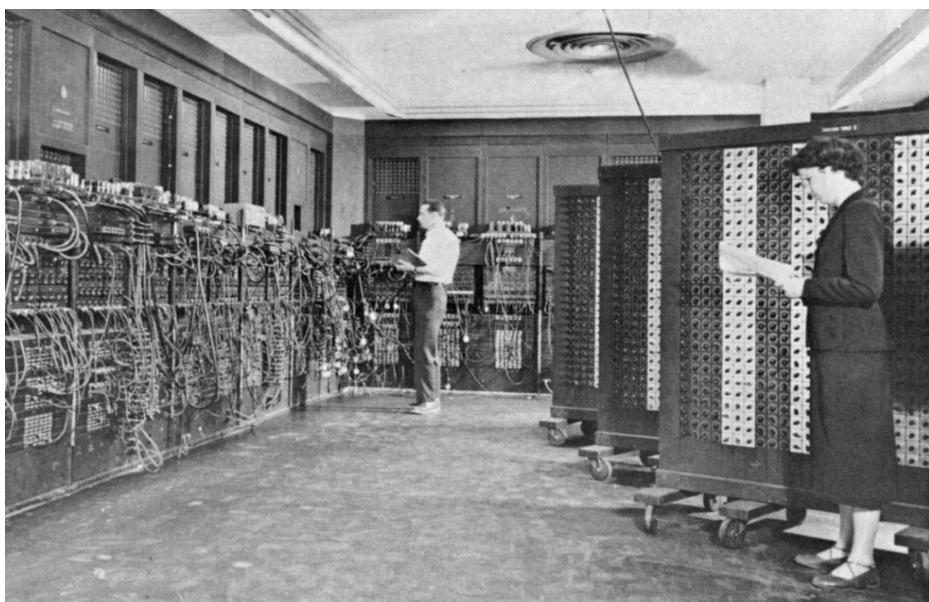


Рисунок 1. Первый американский компьютер «ЭНИАК»

Толстые провода, которые вы видите на фото — это не питание блоков, а ничто иное, как часть программы (вторая часть располагалась на задней стенке и выглядела ещё ужаснее — переплетение штеккерных и паянных проводов 😊). Чтобы запрограммировать такое чудовище, надо его сначала выключить, подождать, пока разряжаются конденсаторы, а потом бегать вдоль лицевой части как угорелому, перетыкать провода и перешёлкивать многотысячные пакетные переключатели. А то ещё, не дай-бог, не дождался разряда кондёров и

схватился за сигнальный конец — прощай семья и дети, здравствуй кладбище...

Несмотря на то, что разработка велась с начала 1943 году, к концу войны доделать её не успели. К испытаниям работы приступили только осенью 1945 года и полностью она отлажена была лишь зимой 1946 года. Правда никто сильно расстраиваться не стал и на «ЭНИАК» стали делать расчёты для атомного проекта который, из-за успешно проведённых испытаний в Хиросиме и Нагасаки, только набирал обороты.

Если честно, считать «ЭНИАК» чем-то выдающимся как-то неохото. Это «произведение искусств» больше подходит под категорию «как не надо делать компьютеры». Вот посудите сами:

Электронная основа триггеров — 17,5 тысяч электронных ламп. Такое ужасающее количество получилось оттого, что машина задумана на основе десятичной системы счисления. Система охлаждения хоть и мощная, но со своей задачейправлялась с трудом. Каждый час перегорало по десятку ламп на поиск которых могло уйти и сутки, и двое. Проблема усугублялась тем, что при включении машины счёт перегоревшим лампам уже мог идти на сотни. Последний недостаток решили довольно оригинально — при выключении снималось только высокое напряжение с сеток и катодов ламп. Напряжение подогрева анодов не выключалось, а просто снижалось с 6.3 вольта до 5.7. Таким образом, машина жрала во включённом состоянии ~ 175 кВт, а в выключенном ~ 158 кВт. Фокус с питанием и оптимизированная система охлаждения позволили выйти на рекорд — в машине стало перегорать «всего» 2...3 лампы в неделю.

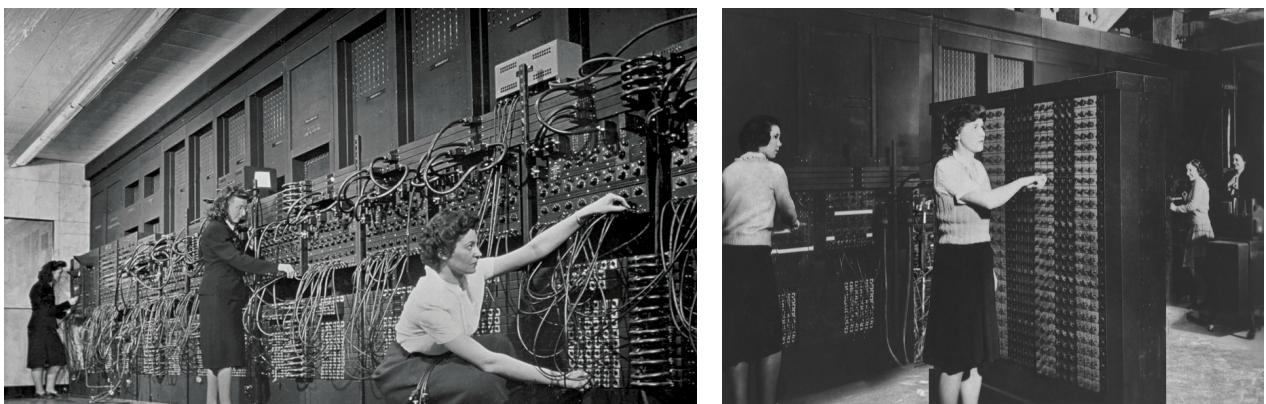


Рисунок 2. Процесс программирования компьютера «ЭНИАК»

Десятичная система счисления конечно же возникла не просто так. Изначально предполагалось, что с машиной будут управляться не особо образованные девицы, которые раньше занимались ручными расчётами.

Естественно, ввод и вывод данных должен быть только десятичный и никак иначе. В противном случае дополнительно пришлось бы пользоваться переводными таблицами из десятичной системы в двоичную и обратно. Как вы понимаете, при этом немедленно бы возросло количество ошибок ввода\вывода данных, что свело бы на нет всё преимущество скорости вычислений. Первое время с машиной работало около 50 инженеров, но когда основная часть техпроблем была решена, с программированием управлялось всего шесть девушек. В день они выполняли ту же вычислительную работу, что раньше делали 200 человек за месяц.

Первой ЭВМ вполне могла бы быть машина немецкого инженера Конрада Цузе «Z3», которая заработала весной 1941 года. Это была его уже третья машина. Первые две он делал по образцу работы тогдашних станков с программным управлением — код программы и её данные были реализованы в виде металлических контактов, а выполнение программы обеспечивал электродвигатель. После того, как он познакомился с инженером-электронщиком Гельмутом Шеером и рассказал ему о своей работе, тот довольно быстро осмыслил идею и предложил её реализовать электронным способом — в виде радиоламп. Засели за разработку. Поскольку Цузе в электронике ничего не понимал, на нём лежала разработка концепции, а Шеер уже занимался практическим воплощением. Подготовив эскизный проект, изобретатели пришли в полное уныние — кроме разной электронной мелочи им требовалось около 2 000 ламп. Надо сказать, что Цузе изначально ориентировался на двоичную систему (он объединил алгебру Джорджа Буля и работу Клода Шеннона о способе реализации двоичных схем на электронных реле), поэтому по сравнению с американцами ламп у них получилось немного.

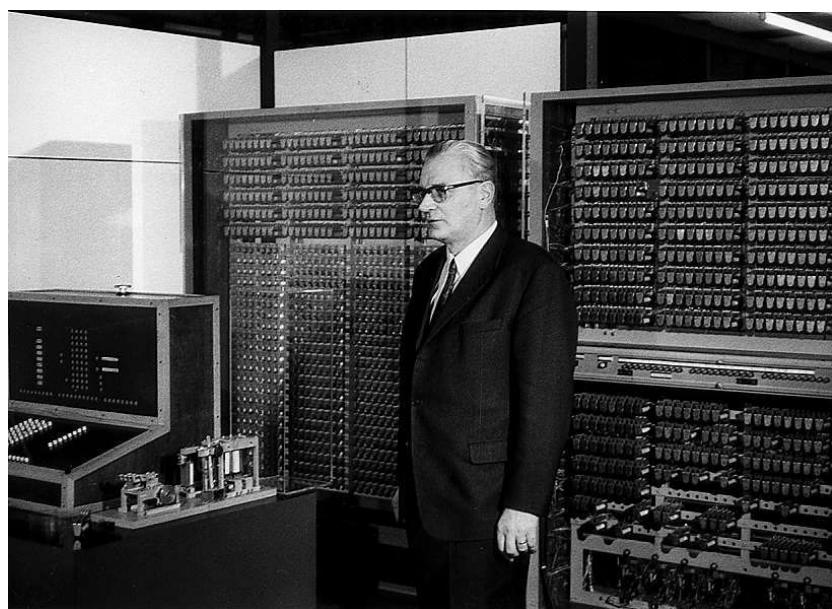


Рисунок 3. Конрад Цузе и его «Z3» (реплика машины 1960-ых годов)

Машина умела делать все арифметические операции и, самое ценное, вычислять квадратный корень.

Лампы, сколько бы их ни было, компонент по тем временам дорогой. Поэтому когда проект был представлен через фирму «Хеншель» военным, те просто не поверили своим глазам. В те времена самым сложным электронным изделием считался магнитофон, который имел на борту 6...8 ламп. 2 000 ламп явно уже ни у кого в голове не умещалось. В деньгах ему отказали, однако проект не закрыли. Через некоторое время Цузе и Шеер представили «облегчённую» версию — ни одной лампы, а всё на телефонных реле. Вот это уже благополучно прошло.

Теперь можно сравнить «ЭНИАК» и «Z3». У последней — два шкафа начинки и пульт управления. И все... ☺ Электричества она потребляла 1 кВт и только когда работала. В выключенном состоянии электричество ей было не нужно. Конечно, её стоит называть не электронной, а электрической вычислительной машиной, но Цузе тут не виноват. Самый смак здесь — программа была не в виде громадного количества проводов, а хранилась в машине, в реле. Загружалась с помощью перфорированной бумажной ленты. Изначально предполагалось программу хранить на фотоплёнке, так как она более прочная. Но опять всё упёрлось в деньги. Да и война уже была на носу — фотоплёнка уже отошла в разряд стратегических вещей.

Полная стоимость разработки «ЭНИАК» составила 6 500 000 долларов, а «Z3» — 50 000 рейхсмарок. ☺

«ЭНИАК» больше никто и никогда повторить не пытался. Американский блин оказался комом. Тем более, что сами американцы сделали максимум для того, чтобы именно этот блин скомкать. Кроме непропорциональной сложности и стоимости, «ЭНИАК» выявил ещё и массу чисто человеческих проблем. В 1944 году военный куратор проекта Герман Голдстайн приглашает в качестве консультанта знаменитого (и тогда тоже ☺) Джона фон Неймана. Тот первое время был в полном восторге, а потом всё больше и больше иронически хмыкал, постепенно знакомясь с этапами разработки. Хотя он своими советами внёс довольно большой вклад в разработку «ЭНИАКА», но концепция «проводной программы» ему совершенно не нравилась. Следующая версия компьютера начала проектироваться ещё до окончания «ЭНИАКА» в конце 1944 года. И вот она должна была быть в полном соответствии с фон Нейманом — как минимум программа внутри машины, а не снаружи её. ☺ Экерт даже нашёл способ реализации такой концепции — подсмотрел у создателей радара. У них использовались стеклянные трубки с ртутью для хранения «неподвижной» информации, т. е. о «несамолётах». И тут случилось страшное — фон Нейман напечатал свое воззрение («Первый проект отчёта о EDVAC»). «EDVAC» — так должен был называться новый компьютер) и отоспал его Голдстайну. Естественно в отчёте он упомянул и о ртутных трубках. Отчёт разошёлся довольно широко, тут постарался Голдстайн. Экерт и другой

генеральный разработчик, Моксли, были поражены в самое сердце, так как теперь нечего было рассчитывать на патенты. Короче все друг с другом разосрались и группу разработчиков сначала покинули Мокли с Экертом, а потом и фон Нейман с Голдстайном.

Как следствие, первыми новый компьютер создали не американцы, а британцы, назвав его «EDSAC». Заработал он у них 6 мая 1949 года. Британцы, естественно, отчёт тоже почитали, причём с большой пользой для себя. Вот это уже был компьютер с принципами работы, которые мы используем и по сей день. Кроме оперативной памяти, у него была и такая интересная штука, как дисплей на электронно-лучевых вакуумных трубках, который сначала применялся в отладочных целях, а потом уже стал использоваться для вывода вычисляемой информации.



Рисунок 4. Первый (британский) компьютер EDSAC с дисплеем на ЭЛТ



Рисунок 5. Дисплей поближе...

Именно благодаря этому дисплею, в 1952 году была написана первая компьютерная игрушка «Крестики-нолики», работавшая в графическом режиме.

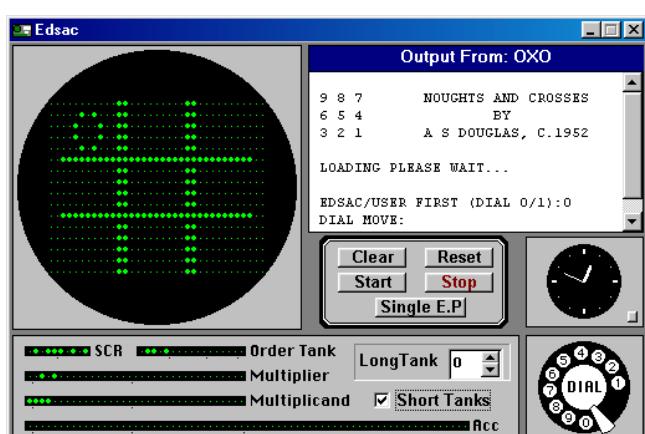


Рисунок 6. «Крестики-нолики» на симуляторе EDSAC

Американский «EDVAC» был предварительно готов к концу 1949 года, однако, как и «ЭНИАК», имел такое большое количество технических проблем, что более-менее нормально заработал только в 1951 году, находясь большую часть времени в режиме отладки. Окончательно принят в эксплуатацию в мае 1952 года.



Рисунок 7. «EDVAC» и команда его инженеров-отладчиков

Наша первая ЭВМ — МЭСМ, выглядела намного культурнее, чем американский ЭНИАК. Что и неудивительно, т. к. за образец частично был взят британский EDSAC. ☺ Проектирование началось в 1948 году, а первая программа заработала уже 1950-ом. Сразу хочу сказать, что МЭСМ отчего-то в наших интернетах расшифровывают неправильно. Довольно часто можно увидеть расшифровку «**Малая Электронно-Счётная Машина**», что совереннейшая неправда. Ведь тогда же должна была бы существовать ещё и Большая, но до Большой было ещё далековато... Правильная расшифровка — «**Модель Электронно-Счётной Машины**», поскольку она была именно моделью, а не настоящей машиной. На ней отрабатывались алгоритмы и технологии построения электронно-счётных машин.

Кстати, в истории создания МЭСМ довольно много интересного. Для работы проекта команде, во главе с профессором Лебедевым, было отведено помещение бывшего монастыря недалеко от Киева. Точнее говоря, это была бывшая монастырская гостиница. По этому поводу шутили, что наша ЭВМ создана с божьей помощью. ☺

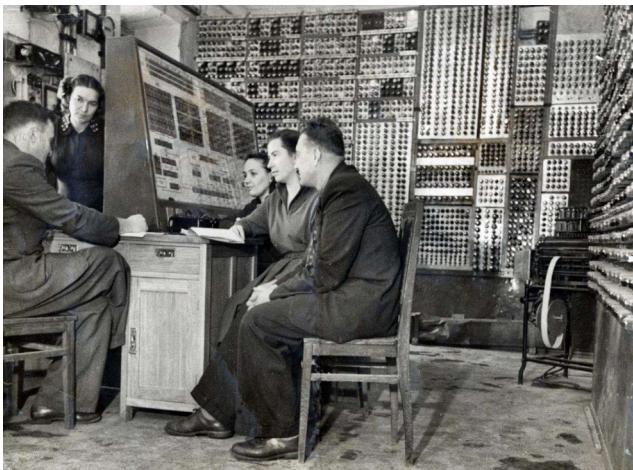


Рисунок 8. Первая советская вычислительная машина — МЭСМ



Рисунок 9. Сергей Алексеевич Лебедев, создатель МЭСМ (а потом и БЭСМ)

Многие считают, что ЭВМ мы слымзили у американцев. Вношу поправку — не только у них... ☺ В этом вопросе нам существенно помогала знаменитая кембриджская пятёрка во главе с не менее знаменитым Кимом Филби. Говорят, когда Ким Филби увидел наш знаменитый сериал «17 мгновений весны», он очень долго веселился по поводу Вячеслава Тихонова в главной роли. По его мнению, с таким не в меру серьёзным лицом он попался бы в первый же день. ☺

Вопрос по поводу «слымзивания» очень тонкий. Не думайте, что если что-то сопрёшь, то этим можно тут же и пользоваться. С электроникой, и вообще с наукой, это далеко не так. В этом убедились и создатели атомной бомбы, и создатели ЭВМ. Идеи — они, в общем-то, универсальны. Подводит только конкретная реализация. Как оказалось, американские схемы ЭВМ собрать на наших компонентах не представляется возможным — это вам не аналоговая электроника. К примеру, разные фронты импульсов у американских и наших компонентов сводят на нет всю интересность американских схем триггеров. Поэтому фактически делать расчёты и разрабатывать схемы пришлось заново.

После того, как машина была полностью собрана и готова к работе, пришла пора её испытывать. Два лучших, не сильно занятых, математика составили сложный контрольный тест. Работали они параллельно над одним и тем же. Два, потому что один может случайно ошибиться, а двое — всё же какая-то гарантия, потому что с разными ответами будет сразу понятно, что где-то закралась ошибка. Составлять тест пришлось в двоичных числах, потому что в то время машины ничего другого просто не понимали.

Ввели программу, нажали кнопку «Старт» и, увы, на одном из этапов машина выдаёт ответ не тот, что был написан на этом же этапе в тесте. Ужас, суматоха, разработчики уже начали прощаться с родными и сушить сухари... Однако Лебедева так просто не возмёшь. Большую часть народа он послал

прозванивать лампы, диоды и конденсаторы, а сам уединился в своём кабинете и начал тщательно проверять тесты, т.к. где-то в глубине своей научной души не верил, что ошибка была именно в машине. Упарились все и прежде чем машину успели разобрать всю до последнего винтика, Лебедев нашёл ошибку в teste. И что интересно — в одном и том же месте ошиблись оба математика. А вот машина посчитала всё верно.

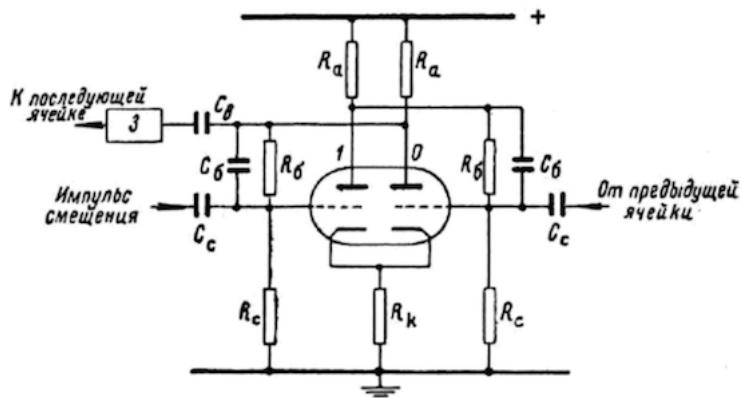


Рисунок 10. Схема элементарной ячейки блока памяти арифметического устройства МЭСМ

Вернёмся к вопросу программирования. 😊 Время шло, появились компактные пульты с переключателями, за которым уже можно было сидеть спокойно. Но легче программисту от этого не стало. Даже сидя набирать машинные команды ручками довольно тяжело, потому что программы усложнялись и команд становилось всё больше и больше.



Рисунок 11. Пульт управления компьютером IBM 704

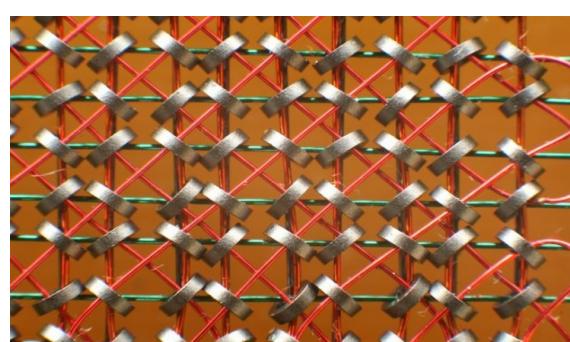


Рисунок 12. Оперативная память 50-ых годов, где «жила» набранная программа

Надо сказать, что повышением удобства работы с компьютером занимались многие, однако, в первую очередь, это касалось, как говорят авиаторы, систем механизации. Дела первое время шли ни шатко, ни валко. Компьютеры тех времён, по нашим понятиям, были очень бедны ресурсами, поскольку электроника тогда стоила очень дорого, что совсем не способствовало резкому увеличению количества ЭВМ, а значит и повышению удобств пользования ими. Но всё меняется — денег становилось больше, производство шло в гору (для США после второй мировой это неудивительно

— догадайтесь почему 😊), повышались скорости процессоров, увеличивалась оперативная память, появились удобные, привычные всем пишущие машинки, которые стали клавиатурами. Появился дисплей для вывода оперативной информации. Не сам компьютер, но система управления им стала больше походить на то, к чему мы сейчас привыкли.



Рисунок 13. Первые дисплеи с графикой

В программировании, в первую очередь, сделали человекообразие для команд процессора, т.к. буквы легче запомнить и понять, чем цифры. И появился язык ассемблера. Википедия пишет, что произошло это событие в 1949 году [3], но на самом деле с точными датами не совсем понятно. Ориентировочно можно считать — 1949 ... 1950 год.

Программировать стало немного легче, но только для самих программистов. Программирование продолжало оставаться жёстко привязанным к определённому аппарату. Таким образом, написав программу для IBM 704, запустить её на UNIVAC не представлялось никакой возможности. Но, лиха беда начало. После ассемблера стало совершенно понятно, что всю самую тяжёлую и неквалифицированную работу по программированию можно переложить на сам компьютер. Вот тут-то можно и начинать историю языков программирования, независимых от железа.

Во второй половине 50-ых почти одновременно появились 3 специализированных языка, которые своим синтаксисом не были привязаны к компьютеру:

1. **Кобол (Cobol)** — COmmon Business Oriented Language, примерно переводится как «простой бизнес-ориентированный язык». Практически всё так и есть. Кобольные программы писались самым обычным (английским) языком. Пожалуй до сегодняшнего времени это самый человекоориентированный язык программирования. Его главная фишка — простая работа с данными и хранилищами данных;

2. **Лисп (Lisp)** — LISt Processing language, язык обработки списков.

Первоначально основным типом данных в нём были списки. Язык предназначался для разработки систем искусственного интеллекта, поскольку в то время любимая фишкой многих была «научить компьютер думать как человек». Правда, несмотря на то, что прошло довольно много времени, даже приблизится к этому не удалось ☺. Но системы искусственного интеллекта вовсю функционируют и помогают человечеству увидеть много чего невиданного ранее;

3. Фортран (Fortran) — FORmula TRANslator — переводчик формул. В отличие от Лиспа, изначально предполагался, как и Кобол в бизнесе, человекообразный решатель научных и прикладных задач с помощью численных методов математики и математической физики.

Фортран я поместил на последнее место не просто так. Дело в том, что он является непосредственным прародителем языка Паскаль. Хотя Кобол и Лисп тоже оказали влияние на появление Паскаля, но всё же большинство взято именно из Фортрана. Кстати, тут свою роль сыграло ещё то, что создатель Фортрана Джон Бэкус, так же как и Никлаус Вирт, по своему образованию инженер, поэтому при создании языка он, как и Вирт, исходил исключительно из практических соображений. Хотя Фортран (как и Паскаль впоследствии) любили поругать (особенно этим отличались дикие апологеты языка Си ☺), но не стоит забывать о практических предпосылках Фортрана:

1. В качестве образца высокоуровневого языка тогда был только ассемблер, поэтому чтобы программисты сильно не воротили свои носы от нового языка, он был похож и на язык ассемблера;

2. Введение чисто человеческих слов (как в Коболе) для описания логической структуры программы и тех математических функций, которая она должна вычислять, позволило непрограммистам быстро осваивать этот язык и самим писать программы, не делясь своим и без того скучным заработка (а учёные во всём мире зарабатывают не так уж и много...) с дорогими (в буквальном смысле ☺) программистами.

Тут же хочу отметить, что одно из основных обвинений в сторону Фортрана, дескать там не надо было объявлять типы данных, из-за чего может возникать куча ошибок. Это обвинение несостоятельно потому, что в Фортране тип переменных определялся компилятором по первой букве переменной. Да, там существовали «умолчальные буквы», чтобы учёным-непрограммистам не тратить много времени на написание «не относящегося» к теме программы кода. Кстати говоря, в современном PHP типы тоже не торопятся объявлять, оставляя это на усмотрение интерпретатора и ничего, никто не плачет по этому поводу... ☺

И вот, на фоне всех этих плясок с бубном появляется, во всей красе и великолепии, Никлаус Вирт...

2. А вот теперь сама история. Вирт и его Паскаль



Рисунок 14. Никлаус Вирт сразу после изобретения языка Паскаль... ☺

Несмотря на то, что изобретателем Паскаля везде записан Никлаус Вирт, хочу вас немедленно разочаровать. Оговорюсь, никаких страшных тайн тут нет, но вот время одиночек давно прошло. Даже в стародавние времена, когда, к примеру, Витус Беринг открыл пролив имени себя, открыл он его совсем не один, а в составе руководимого им экипажа корабля. Понятно, что в одиночку он туда ни за что бы не добрался, а значит и открытия бы никакого не было. Аналогичная история и с Виртом. Да, заслуги его несомненны, но без помощи своих соратников, помощников и друзей, у него могло ничего бы и не получиться.

То, что Вирт вырос таким гениальным — несомненная заслуга родителей. Гены здесь играют хоть и важную, но не основную роль. Родители его правильно воспитывали. Причём оба — и мать и отец. Сначала ни о каком программировании речь не шла. В те времена профессии такой просто не было. Была тяга к технике, как и у большинства мальчишек. Отец его в этом всемерно поддерживал. С детства Никлаус мечтал стать изобретателем самолётов. Как вскоре выяснилось, без вычислений, причём серьёзных, никаким изобретателем тут не станешь. К моменту поступления Вирта в высшую техническую школу в Цюрихе (ETH), интересы его сместились в область управления летательными аппаратами. Видимо поэтому была выбрана специальность электроника, что повторяет историю другого великого изобретателя языков — Джона Бэкуса. Надо сказать, что в Цюрихе Вирт получил только степень бакалавра электротехники (электроника — это очень заумная часть электротехники ☺). Теоретически можно было бы сразу поступать и в магистратуру, дипломные и

экзаменационные оценки этому нисколько не препятствовали. Однако, в магистратуру ETH можно поступить только если найдётся персональный наставник по выбранной специальности. Увы, для Вирта такого наставника не нашлось. Впрочем, поползновения на магистратуру у него были чисто теоретические, т.к. у родителей денег на неё всё равно бы не хватило.

После учёбы Никлаус пошёл работать инженером-электронщиком в одну из фирм Цюриха. И очень быстро понял, что сидеть 8 часов за столом, пытаясь усовершенствовать какой-то усилитель, ему как-то не очень хочется. Терпения хватило только на месяц, после чего он уволился. Если бы дело происходило в СССР, то пришлось бы сильно скандалить, поскольку без отработки 3 года после учёбы его никто бы не уволил. Но Швейцария страна тихая, все там учатся за свой счёт, поэтому бывшие студенты, если денег у родителей нет, должны только банку. Никлаус никому должен не был, так что его уволили без всяких возражений.

Очень хотелось науки. Вот только с повышением квалификации в Цюрихе оказалось совсем туго. Нет, можно было конечно взять кредит в банке, но это уже должны бы сделать родители, поскольку самому Вирту, без работы и с отсутствием стажа, денег никто не даст.

Надо сказать, что поехать в Америку уже тогда было любимым делом многих молодых людей из Европы. Однако решиться на такое очень непросто. Отец был не против чтобы сын «повидал мир», в том числе и в финансовом плане, против была мама. Вдобавок Вирт буквально только что женился, поэтому переезд чёрте куда за море был делом совсем призрачным. Помощь пришла неожиданно. Его старший приятель три года назад уехал в Нью-Йорк и вроде там неплохо устроился. Так вот, он сказал, что Вирт с женой может первое время перекантоваться у него.

И вот куплен билет на пароход, пролиты слёзы прощания, здравствуй Гавр, 7 дней очень скучного плавания в 3-ем классе — и Вирт в Нью-Йорке. Как оказалось, вновь прибывшим в страну мечты можно устроится на работу только либо мыть посуду, либо мыть туалеты. Грузчик уже считалось профессией очень престижной. Совершенно случайно Вирт узнаёт, что в университете Лаваля, Квебек, организована новая кафедра, куда набирают студентов. Поскольку про кафедру никто не знает, народ туда не сильно то и рвётся. Узнав, что там же можно получить и работу (скажем честно, учёба в Лавале — удовольствие не из дешёвых), а вдобавок студентам-отличникам можно получить солидную скидку, Вирт едет туда и без проблем поступает в магистратуру.

Что интересно, университет Лаваля изначально создавался как религиозное учебное заведение. Однако там быстро поняли, что без светской науки далеко не уедешь. Хотя религия из универа совсем и не ушла, но большинство направлений теперь совершенно светское. Религия осталась только как научная дисциплина «Теология и религиоведение».

Результатом учёбы стала магистерская диссертация, и пробуждение

интереса к программированию. Как отличник, Вирт получил возможность поступить в университет Беркли и там попал в руки знаменитейшему профессору Гарри Хаски. Хаски, кроме того что был отличным математиком, являлся первым американским компьютерщиком, участвовавшим ещё в создании Эниака.

Из интересного. В семье Хаски кого либо с образованием, даже средним, никогда не водилось. Однако его отец пользу учения видимо понимал хорошо, поскольку оплатил сыну сначала колледж, а потом и университет. По всей видимости мальчик Гарри был гениален с самого рождения. И, вдобавок, везуч. После получения степени доктор философии, его сразу же взяли в Пенсильванский университет, где он по контракту с ВМФ США читал местным морякам лекции по математики. Моряки его очень уважали за умение доступно объяснять сложные вещи. Прожил Хаски 101 год, несмотря на то, что был большой весельчак и не дурак выпить. И, добавим, большой любитель женщин. Второй раз он женился в 78 лет. ☺

В Беркли Вирт, кроме написания диссертации и получения учёной степени доктора философии (Ph. D, по нашему кандидат наук), изобрёл новый язык программирования — Эйлер, разработанный в соавторстве с Гельмутом Вебером. Хотя по официальной версии этот язык является всего лишь усовершенствованной и расширенной версией Алгол-60, однако расширен он был в результате анализа других языков — в нём явно просматриваются как следы Лиспа (списки), так и следы Фортрана (динамическая типизация). Кстати говоря, сама разработка велась по большей части на Лиспе.

Эйлер начинался с попытки разобраться с кодом Алгола, который тогда ещё был очень предварительным (Алгол-58) и о существовании которого в США узнали от Джона Бэкуса, принимавшего участие в весьма бурных дискуссиях по этому новому европейскому языку программирования. Команда Хаски, которая состояла из таких же аспирантов, как и Вирт, мало что могла понять, потому что хотя Алгол и был слегка похож на популярный в США Фортран, но всё же не до такой степени, чтобы в нём разбираться с первого взгляда. В конце-концов всё-таки удалось понять о чём шла речь в коде программы. Вирт и Гельмут решили, что чем так мучатся, лучше всего написать свою версию Алгола. Не стоит думать, что это какое-то экстраординарное решение. С Фортраном, до его стандартизации, было то же самое — каждый университет считал своим святым долгом сделать себе такой транслятор Фортрана, какой считал для себя удобней.

Примерно в 1964 году была сделана практическая реализация Эйлера для местной ЭВМ, сначала для IBM 704, а потом и для более новой IBM 360. В дальнейшем следы этого языка теряются... Уже в Эйлере стало понятно, в каком направлении Вирт усматривает развитие языков программирования — простота реализации.

Диссертация Вирта произвела большое впечатление на учёный мир. В те времена даже простой программист был большой редкостью. А уж

программист с научной степень и, тем более, создавший новый язык... Вирта приглашают в комитет по стандартизации языка Алгол, для разработки третьей (как выяснилось позже — последней) версии этого языка. Ещё до прихода Вирта явно обозначились два направления в развития языка:

1. Группа, где учёные разделяли взгляд Вирта на модернизацию языка, когда язык должен быть именно модернизирован, дабы не мучатся с изучением совсем нового неизвестно чего. И, естественно, чем проще — тем лучше. А уж в дальнейшем на этом языке писать дополнительные модули с дополнительными фишками. Этот вариант языка условно назывался *Algol-X* (от слова «*eXtended*»). В группе иксов были в подавляющем большинстве практики.

2. «Группа искусственного интеллекта», где разрабатывалась концепция всеобъемлющего языка, на котором можно сделать абсолютно всё и сразу. И здесь собрались чистые математики, поклонники цифры и алгоритма (в чистом виде, разумеется ☺). Этот вариант условно назывался *Algol-U* (от слова «*Universal*»).

Надо сказать, что вторая группа имела в комитете значительную поддержку. Дело в том, что с самого начала язык Алгол создавался как европейская альтернатива американскому Фортрану. И можно сказать, что это удалось, по крайней мере для версии стандарта Алгол-60. Тем более что Алгол создавался уже на основе опыта работы с Фортраном и все (почти) синтаксические недоработки Фортрана были учтены. Примерно в 1964 году в США появился универсальный «всё-в-одном» язык PL/1. Кстати говоря, Вирт тоже поучаствовал в его разработке. Так вот, в Европе опять захотели переплюнуть американцев со следующий версией Алгола, тем более, что в прошлый раз получилось вполне себе ничего. Увы, увы... PL/1 в Америке довольно быстро заглох, даже несмотря на титанические усилия IBM по его продвижению. В 60-е годы компьютеры уже имели впечатляющую производительность. Проект Сэймура Крэя был способен выдавать на гора миллион, а то и больше операций в секунду. Правда крэевские компы довольно серьёзно грелись, но эта проблема решаема. И вот несмотря на такую мощь, ресурсов компа на полноценную реализацию PL/1 всё же не хватало. Алгол пошёл по тому же пути. Проект *Algol-W*, предложенный Виртом в соавторстве с Дейкстрой и Хоаром, комитетом был отклонён. Уж даже и не знаю к добру или худу, потому что если бы его приняли, Паскаль бы точно не появился. Но мы могли бы до сих пор работать на Виртовском Алголе. Высокоинтеллектуальный Алгол-68 постигла та же участь, что и PL/1. Несмотря на то, что держался он намного дольше, даже был в СССР, ресурсов компьютеров на 100% реализацию было маловато. Обычно на его основе делали компиляторы, которые поддерживали стандарт частично с той или иной степенью полноты. Собственно проблема была даже не в его внешней

сложности, а в том, что разрабатывая компилятор на основе описания языка, можно запросто провалится в «чёрную дыру». Это буквально в двух словах выражено в [4, стр. 163]. Тут надо сказать, что влияние Вирта видно даже здесь, поскольку изучение языка таких уж непреодолимых трудностей (согласно [4, стр. 163]) не вызывало.

В создании первоначального варианта Алгола принимал активное участие Джон Бэкус. Несмотря на то, что «научные споры» по поводу нового языка больше напоминали свару в базарный день, потому что каждый считал себя правым, а американская часть делегации ещё и усилено продвигала идею Фортрана, как основы «глобального компьютерного языка», для Бэкуса это явилось поводом и идеей написать «язык описания языка». И в результате мы имеем «нормальную форму Бэкуса» (БНФ), которую чуть попозже дополнил Питер Наур (кстати говоря, тоже слегка пострадавший от алгольной дискуссии 😊).

Наконец-то про Паскаль...

Уйдя из комитета Алгола, как следует хлопнув дверью, Вирт решил последовать примеру Владимира Ильича Ленина, который сказал «Мы пойдём другим путём». Правду нужно доказывать делами, а не словами.

Впервые Паскаль, как язык на основе не принятого Algol-W, появился в 1968 году. Именно в этом году Вирт со своими помощниками подготовил предварительное описание языка, где термин «Паскаль» и был применён. Примерно 2 года он находился больше в области теории, чем практики. Уточнялись отдельные концептуальные моменты, дорабатывались, и теоретически и практически, отдельные узлы языка, проверялось, как на практике будет выглядеть реализация того или иного языкового постулата. Теория постоянно проверялась практикой на имевшемся в наличии PDP-11. Надо сказать, что Вирт к этому времени уже переехал в Цюрих. Сначала его приглашали на кафедру Цюрихского университета (который находился буквально в двух шагах от ETH). Вирт обещал подумать, поскольку он к тому времени уже работал в Стенфорде, а переезд на новое место жительства дело отнюдь не плёвое. К тому же ничего нового в Цюрихе ему не светило, так что менять шило на мыло... Ну, разве что, жить поближе к родителям...

Буквально через месяц или два ему пришло приглашение из альма-матер — ETH, где собирались организовать новую кафедру — информатики. И поскольку Вирт не просто знаменит, а буквально свой человек, руководство университета никого кроме Вирта завкафедрой не видело. Тут уж Вирт не стал долго раздумывать и отправился заведывать...

В 1970 году высшей технической школой Цюриха, где работал Вирт, был закуплен один из наиболее мощных крэевских компов — CDC 6000-ой серии. К сожалению программ для него оказалось маловато. Вот тут-то Вирт и оказался в своё время и на своём месте. Многие (начитавшись ругательных отзывов о Паскале, которые писали сишиники 😊) продолжают повторять байку прошловековых 70-ых годов, что дескать Паскаль предназначался исключительно для учебных целей, а никак не для нормальной работы. Ничего подобного. В 1970 году, как раз для этой новой CDC был написан компилятор Паскаля. Полное описание языка появилось в 1971-ом. А вот учебно-методическая литература и курсы на её основе появились только в 1972-ом году. Надо сказать, что Паскаль специально подгонялся под этот компьютер, поэтому тогдашние ограничения в Паскале объясняются именно особенностями CDC, а отнюдь не принципиальной позицией Вирта по этому поводу.

В Европе в то время очень активно продвигался Алгол, как основной алгоритмический язык. Казалось бы проект Вирта, у которого не было мощной финансовой поддержки, был обречён. Но не тут-то было. Повторилась история Фортрана. Паскаль внезапно оказался легко реализуем на самых разных

компьютерах. И особой изюминкой было то, что синтаксически он не сильно отличался от текущего Алгола, т.е. человек, учившийся языку Алгол, мог с первого взгляда понять текст программы на языке Паскаль. Для примера:

Язык Алгол	Язык Паскаль
<pre> procedure matmul(m, n, a, b) result:(c); value m, n; integer m, n; real array a, b, c; begin integer i, j, k; for i:=1 step 1 until m do for j:=1 step 1 until n do for k:=1 step 1 until n do c[i,j]:=c[i,j]+a[i,k]*b[k,j]; end matmul; </pre>	<pre> Type TMatrix = array[1..20, 1..20] of real; procedure matmul(m, n: integer; a, b: TMatrix; var c: TMatrix); Var i, j, k: integer; Begin For i:=1 To m Do For j:=1 To n Do For k:=1 To n Do c[i,j]:=c[i,j]+a[i,k]*b[k,j]; End; </pre>

Здесь представлены процедуры умножения матриц. Как видим, и тот и другой код опознаются и соотносятся друг с другом очень легко. Здесь только два кардинальных различия и одно кажущееся:

1. Описание переменных и параметров процедуры. В Алголе описание переменных и параметров унаследовано от Фортрана. Но чтобы выделить описание параметров, оно сделано под заголовком, ещё до «begin». Служебное слово **value** означает передачу параметров по значению;

2. Цикл с известным количеством повторений (for). В Алголе, так же как и в Фортране, такой цикл допускал установку произвольного шага. В Паскале шаг подобного цикла мог быть равен только 1 или -1. В случае другого шага пришлось бы менять тип цикла на **Repeat** или **While**, что не является такой уж проблемой;

3. Массивы. Может показаться, что Алгол спокойно работает с безразмерными массивами, чего в Паскале не было. Однако это не совсем так. В Алголе тоже, как и в Паскале, нужно предварительно задавать размер массивов. Однако массивы в Алголе, как и в старом Фортране — ребята «гулящие». Тут главное не выйти за рамки фактического размера, а вот его описательный размер, но только в плане передачи этого массива в процедуру, мог быть написан от фонаря. Т.е. если говорить в общем, то размер массива описывается только в главной программе. В процедуру он передаётся по ссылке, а значит его значения никуда не теряются, если в параметрах вместо A[10] будет описан A[1]. Кстати говоря, в старом Фортране именно так любили описывать в подпрограмме массив с неизвестным заранее размером. Фактически в Алголе была приведена в более-менее внятный вид форTRANья практика работы с массивами при передаче их в подпрограмму.

Эти различия совершенно не мешали понимать паскалевский текст. Мало того, довольно быстро выяснилось, что при наличии на вычислительной машине альтернативы Алголу, в виде Паскаля, приводило к тому, что больше половины работающего там народа предпочитали именно Паскаль. Многие даже переводили тексты алгольных программ в паскальные. Надо сказать, что в своё время такая в высшей степени интересная организация, как Ассоциация вычислительной техники (Association for Computing Machinery — ACM) [5], несмотря на свою американскую штаб-квартиру, предпочитала публиковать алгоритмы именно на языке Алгол, хотя с Фортраном они там тоже дружили. Таким образом, учёный народ Цюриха, получив свеженький номер журнала, например, «Transactions on Mathematical Software» [6] с новыми алгоритмами, себя не помня от восторга кидался в машинный зал ВЦ (в прыжке, которому позавидовал бы Майк Пауэлл) грудью падал на ближайший терминал, чтобы никто из коллег не опередил, и немедленно, забыв про обед и сон, начинал перебивать код с Алгола на Паскаль.

Тут честно надо сказать, что по коду у Паскаля, по сравнению с Алголом, каких-то прямо таки уж неоспоримых преимуществ не было. Да, Паскаль был немного попроще, но с другой стороны Алгол был явно поуниверсальнее (возвращаясь к циклу **for**, например). Принципиальная разница начиналась на уровне компилятора. Компиляция кода Паскаль происходила очень быстро. И это было бы преимуществом не решающим, если бы не ещё один, уже сильно печальный недостаток компиляторов Алгола — неполнная реализация стандарта. И ладно бы просто неполная, но вот компиляторы на разных машинах поддерживали разный набор функционала из стандарта Алгол-68. И если более ранний Алгол-60 поддерживался компиляторами на уровне можно сказать 100%, то с Алголом-68 разброс был порою очень серьёзным. В то время как функционал Паскаля был реализован на 100% того, что описал Вирт.

Несмотря на то, что Паскаль планировался как язык, не привязанный к конкретному железу, полной «отвязки» сначала добиться не удалось. В качестве примера можно привести такую интересную штуку, которую современные программисты вряд ли смогут объяснить. ☺ Даже в наиболее поздней книге Вирта по Паскалю (например, [7]) и, что ещё удивительнее, в постсоветских книжках (например, [8]) можно встретить такую конструкцию:

```
program MyProg(INPUT, OUTPUT)
Begin
...
End.
```

Что такое эти таинственные «INPUT» и «OUTPUT» в принципе можно догадаться и без героических мозговых усилий. А вот на кой они тут вообще нужны — вряд ли кто-то сейчас взяточно ответит. А дело было так... Понятно, что INPUT и OUTPUT — это некие стандартные умолчальные устройства ввода

информации и её вывода. Для нас нынешних такими умолчальными устройствами являются клавиатура и экран монитора. А вот для тех легендарных времён всё было не так однозначно... И мониторы были большим дефицитом, поэтому стандартным устройством вывода можно было считать либо АЦПУ (Автоматическое Цифровое Печатающее Устройство — да-да, весь информационный мусор ложился по большей части на бумагу), либо устройство хранения на магнитной ленте. Так же и с устройством ввода — например дырявые карточки (перфокарты) или та же магнитная лента. INPUT и OUTPUT — это стандартные переменные окружения командной оболочки или операционной системы. Именно там определялось, что будет скрываться за стандартными устройствами ввода и вывода. Именно поэтому в программе их надо было указывать в обязательном порядке. Вместо INPUT и OUTPUT можно было указать имена файлов.

Что интересно, древние книжки по Паскалю мы можем читать без малейшего труда, точно так же как и использовать написанный там древний паскальный код. Синтаксис Паскаля практически не поменялся с 1970 года. Даже эти самые пресловутые INPUT\OUTPUT можно использовать. Для Бейсика или Фортрана, например, такое уже не прокатывает, там синтаксис изменился очень сильно. И надо заметить, не без влияния Паскаля. Вредные и хвастливые сишики тоже могут повякать, что ихний синтаксис не менялся с 70-ых. В каком-то смысле они правы. Однако правы только для самых наипростейших программ, типа HelloWorld. Если взять что-нибудь посложнее, то там, кроме заголовочных файлов, которые надо искать неизвестно где и это была бы ещё не такая уж и большая проблема, мы столкнёмся с так называемым препроцессором, который нас будет чуть ли не на каждой строчке тоже посыпать неизвестно в какие заголовки. Сами сишики любят критиковать старый Фортран или старый Бейсик за наличие там GOTO, типа у них такого нет. Ну, во-первых, есть и очень много. Чтобы убедиться — можно взглянуть на исходники современных версий MPICH или OpenMPI, где этих GOTO навалом. А во-вторых, вот этот самый препроцессор эмулирует тоже самое GOTO, только сишики стыдливо этот способ почему-то в качестве «ИДИ НА...» предпочитают вообще не замечать.

Кроме собственно языка Паскаль, Вирт довольно скоро начал понимать, что проблемы в работе программ кроются не только в неправильном программировании, но и в той среде, где выполняется программа. Условно говоря, форма и содержание должны соответствовать друг другу. Поскольку программисту добраться до ОС порою очень сложно, на основе аналогичных работ по Лиспу была разработана менее сложная виртуальная машина, которая интерпретирует некий промежуточный код, который выдаётся компилятором. Данная система получила название Pascal-P. В комплекте с виртуальной машиной шёл компилятор, которых всего было 4 штуки (P1 ... P4). Была и пятая версия, но ею занимался уже не Вирт. Кстати, эта система в 90-ых годах послужила основой для разработки всем известной виртуальной машины Java.

Критика языка Паскаль

Возможно кому-то покажется странным, что такой великолепный язык как Паскаль кто-то может критиковать. Но это только на первый взгляд. Таким великолепным Паскаль стал уже к сегодняшнему дню, в том числе и благодаря конструктивной критике.

Может показаться, что Вирт на критику не реагирует вообще, т.к. в интернете практически невозможно найти какие-то документы, где бы Вирт отвечал своим оппонентам. Но если присмотреться к источникам критики и понять её мотивацию, то выясняется одна довольно таки любопытная вещь — 99% критики идёт со стороны людей, тем или иным образом завязанным на язык Си. И что ещё интереснее, концепция критики в подавляющем большинстве случаев — вывести Паскаль из игры, как коммерческого конкурента языка Си, путём утверждений, будто в Паскале что ни возьми — всё плохо... 😊 Опять же, подобная критика лилась в основном с американского континента. Это поневоле заставляет подозревать исключительно коммерческую, а не научную основу критики. Вирт, наученный горьким опытом работы в комитете АЛГОЛ, где американская часть делегации всеми правдами и неправдами продавливала Фортран и, как минимум, американский стиль в Алголе, прекрасно понимал, что ввязываться в подобные споры совершенно бессмысленно. Тем не менее Вирт был очень внимателен к конструктивной критике и вводил в концепцию новых Паскалей то, что он считал необходимым. Например, указатели, которых в первой версии Паскаля не было.

Ещё одна очень интересная вещь, которую можно условно отнести к критике — это появление расширенных описаний Паскаля, которые писал не Вирт или его ученики (например, [9]). Вызвано это тем, что некоторые термины, относящиеся к языку, Вирт в своём описании не расшифровывает, считая их интуитивно понятными. У многих это вызвало желание внести свою лепту в Паскаль. Несколько это удалось — сказать невозможно, т.к. реакции Вирта, как и в первом случае, нет никакой.

От себя хочу добавить, что одной из более-менее интересных статей, критикующей Паскаль, является статья Кернигана «Почему Паскаль не является моим любимым языком программирования» [10, стр. 235]. Хотя я во многом и не согласен с автором, но читать её интересно, в отличие от многих других. И не надо забывать, что критика там приведена для Паскаля начала 70-ых (на год издания книги не смотрите, это просто сборник статей... 😊).

Стандарты Паскаля

В отличие от Фортрана, у которого международных стандартов было аж 7 штук (Fortran-66,77,90,95,2003,2008,2018, по годам выпуска), у Паскаля международных стандартов всего два:

- ISO 7185:1983 [11] — самый первый стандарт, который сделан на основе Виртовского модифицированного наиболее позднего описания, где он специально вносил уточнения и исправления, как раз сделанные по требованию ISO. В настоящий момент этот стандарт отменён, поскольку существует новый;
- ISO 7185:1990 [12] — действующий стандарт, последний раз редактировался в 2008 году. Текст можно почитать здесь [13].

В СССР так же был подготовлен стандарт Паскаля — «ГОСТ 28140-89. Системы обработки информации. Язык программирования ПАСКАЛЬ», который был сделан на основе ISO 7185:1983. Увы, стандарт появился очень поздно — в 1989 году, введён в действие в 1990-ом, поэтому просуществовал недолго и после распада СССР был отменён.

Не открою большую тайну если скажу, что текст стандарта интересно читать только в том случае, если вы планируете изготовить собственный транслятор. В отличие от Виртовского, стандарт ISO имеет целых 98 страниц, но написан довольно просто и толково. Читать его — одно удовольствие. У Алгола, в этом отношении, стандарт более толстый и тягомотный... ☺

В отличие от стандарта языка Си, который описывает только базовые возможности языка, у стандарта Паскаля есть несколько обязательных встроенных функций: 8 арифметических, 2 для округления чисел и 7 общего назначения. Если сравнивать с Фортраном, который просто битком набит разными функциями, у Паскаля всё скромно. Впрочем, с Фортраном ситуация иная, он с самого начала планировался как расчётный язык, поэтому все математические функции продолжают оставаться частью его языка. На сегодняшний день, функции Паскаля лежат не в компиляторе, а в отдельном модуле SYSTEM. Но поскольку этот модуль всегда подключён по умолчанию, то можно спокойно считать стандартные функции частью языка.

Надо сказать, что под термином «Стандарт» в отношении Паскаль чаще всего имеются в виду именно Виртовские описания языка, а вовсе не ISO. За всё время, пока Вирт занимался именно Паскалем (с 1968 по 1974 год плотно, а примерно до 1980 ... 1981 года уже только при наличии свободного времени), он подготовил несколько описаний, в том числе и учебники, часть из которых были переведены на русский язык. Вот некоторые, наиболее популярные:

- «Паскаль. Руководство для пользователя и описание языка» — учебник языка. Книга несколько раз переиздавалась. Паскаль во всех изданиях представлен в версии для CDC-6000, но никаких неудобств в понимание кода и выполнения описанных там программ это не вносит. Примеров очень много, практически все в законченном виде;
- «Алгоритмы и структуры данных» — учебник по алгоритмам. Есть несколько версий — с примерами на Паскале и Обероне;
- «Алгоритмы + структуры данных = программы» — учебник программирования. Похож на предыдущий. Так же для нескольких языков — сначала для Паскаля, а потом и для Оберона;
- «Построение компиляторов» — учебник именно по построению компиляторов. ☺ Естественно есть на примере Паскаля и, в более поздней версии — на примере Оберона.

Кроме этих книг есть ещё ряд монографий, например «Систематическое программирование. Введение», которые носят более фундаментальный характер, но программные примеры в них, как нетрудно предположить, написаны на Паскале.

Практически все книги Вирта на сегодняшний день официально в открытом доступе. Многие можно взять здесь [14].

Этапы трудовой биографии Никлауса Вирта

Кратко об этапах трудовой биографии Никлауса Вирта (или Святого Ника, как его прозвали студенты ☺):

- **1962...1965** — язык Euler (Эйлер). Учёба в университете Лавалля (магистратура) и калифорнийском университете в Беркли (доктор философии (Ph. D));
- **1964...1967** — язык ALGOL-W. Работа в комитете по модернизации Алгола (IFIP Working Group 2.1);
- **1968...1972** — язык Pascal (Паскаль). Начало работы в ETH;
- **1970...1971** — Venus, система разделения времени по пользователям для только что приобретённого ETH мэйнфрейма CDC Cyber;
- **1972...1974** — Pascal-P, виртуальная машина, которая позволяет очень сильно упростить трансляторы программного кода и упростить разработку трансляторов для разных платформ. В последующем стала называться «P-machine». Что интересно, скорость выполнения паскалевских программ на такой P-machine не уступала тогдашнему Фортрану, который по скорости шёл почти на уровне Ассемблера;
- **1973...1976** — язык Modula (Модула). Святой Ник замахнулся на параллельное программирование ☺ ;
- **1977...1981** — система Lilith, «наш ответ американским жмотам!» ☺
- **1977...1980** — язык Modula/Modula-2 (Модула-2). На самом деле это не отдельный проект, а составная часть Lilith, который выделился в самостоятельную часть только после окончания работ над «железом», микропрограммами и ОС;
- **1980...1982** — Computer-Network. Как и Модула, это была составная часть Lilith. На основе спецификации Ethernet для Lilith был разработан сетевой интерфейс и рабочие станции могли общаться друг с другом на скорости до 3 МБ/сек. ;
- **1982** — Laser Printer. Опять составная часть проекта Lilith. Разработан драйвер для принтера Canon LBP-10, который, в отличие от Xerox'овских, рабочую станцию не подвешивал. ☺ Немного позже появилась система предпросмотра уже сформированных страниц;
- **1983...1985** — Modula-2 Compiler. Переписанный с нуля транслятор для Lilith. Эта система уже позволяла получить исполняемый машинный код для конкретного процессора без использования промежуточных виртуальных машин. Что интересно, компилятор тогдашней Ады собирался из исходников полчаса, а в случае Модула-2 — менее чем за 2 минуты ☺ ;
- **1984...1992** — Ceres [15], второй, после Lilith, «суперкомпьютер в кармане». ☺ С самого начал проектировался на однокристальном

микропроцессоре «повышенной доступности», другими словами оптимальным по критериям дешевизны и производительности. В результате испытаний был выбран 32-ухразрядный NS32032 от National Semiconductor. Вместе с ним использовалась 64 штуки микросхем динамической памяти по 256 кб. и отдельно видеопамять — 32 штуки двухпортовых 64 кб. только что появившихся VRAM;

- **1986...1990** — Oberon Language and System — и опять «наш ответ XEROX'у!», ихнему Cedar OS. ☺ Это часть проекта, но уже Ceres. Сначала планировалось использовать в нём Модула-2, однако даже самому Вирту стало понятно, что Модула и сложновата, и не позволяет быстро делать расширения языка;

- **1991...1996** — Oberon-2;

- **1990...1999** — проектирование Программируемых Логических Интегральных Схем (по-нашему ПЛИС, по-американски — FPGAs). ПЛИС — это однократно программируемые логические интегральные схемы. Физическая замена микросхем мелкой логики. Можно сказать — тначало микроконтроллеров. Отличие их от микроконтроллеров в том, что микроконтроллеры обычно перепрограммируемые, а ПЛИС — нет, что-то вроде сочетания процессора и ПЗУ. С ПЛИСами я впервые познакомился, когда купил себе домой советский IBM-совместимый компьютер в 1991 году «Электроника МС 1502». Откровенно говоря я не сразу поверил, что он «совместимый», настолько он был маленький. Дело в том, что на работе к нас был ЕС-1840 военного исполнения, который весил больше 100 кг (точно не знаю, но тащили мы его вчетвером... ☺) и мог выдерживать прямое попадание 100-мегатонной ядерной боеголовки. ☺ «МС 1502» имел размеры чуть побольше современных 17-тидюймовых ноутбуков и весил примерно столько же. Так вот, внутренности у него были не на мелкой логике, как у оригинала или «ЕС-1840», а на ПЛИСах, точнее говоря, на БМК. БМК от ПЛИСов отличались технологией программирования. ПЛИСы программировались как и ПЗУ, т.е. специальным программатором, сигналы на который подаёт компьютер, согласно заранее составленной программы. БМК программируется на заводе с помощью специального технологического трафарета, т.е. он заточен под массовый выпуск на конвейере. А вот работа и у ПЛИС и у БМК идентична — согласно заложенной программе.

Вирт и его помощники создали систему программирования ПЛИС в составе:

- Язык описания логики Lola (Почитать про язык можно здесь [16]);
- Компилятор с этого языка в некую промежуточную структуру данных для возможности дополнительной обработки;
- Редактор схем, которые должны закладываться в ПЛИС(FPGAs);

➤ Инструменты проверки готовой прошитой ПЛИС с исходной программой на Lola, т.к. сами понимаете, прошивка не всегда идёт гладко из-за возможных косяков внутри микросхемы.

- **1995...?** — в качестве практического применения вышеописанного проекта Вирт создал бортовое ПО для беспилотного вертолёта на ARM-процессоре, совместно с институтом измерений и контроля;
- **1999...2007...2013...?** — Oberon-07.



Рисунок 15. Электроника MC 1502 с монитором тоже Электроника



Рисунок 16. Клавиатура вполне похожа на IBM



Рисунок 17. Ceres-2, рабочая станция на микропроцессоре.
Вид снаружи и изнутри



Рисунок 18. Ceres-3, рабочая станция на микропроцессоре.
Вид снаружи и изнутри

В 1981 году к Вирту и разработке Lilith присоединился профессор Юрг Гуткнхт.



Рисунок 19. Профессор Юрг Гуткнхт, разработчик ОС для виртовских компьютеров и соавтор Оберон-07

Разработка ОС для Lilith и Ceres — это полностью его заслуга. В дальнейшем он более плотно занимался и языком Оберон, а последние две ветки Оберона — Active Oberon и Zonnon, уже полностью его проекты. Из интересного. Как-то у Юрга в интервью спросили:

- Почему Вы занялись Обероном?
- Исключительно потому, что мне не нравился Си++ — ответил он. ☺

С 1988 по 1994 год Вирту Обероном помогал заниматься Ханспетер Мёссенбёк. В Обероне-2 есть очень большая часть работы Мёссенбёка.



Рисунок 20. Профессор Ханспетер Мёссенбёк, соавтор Оберон-2

Вирт несколько раз приезжал в Россию и добрался даже до Томска и Новосибирска. Не могу не сказать, что дядька (а назвать его дедушкой язык не поворачивается) он чрезвычайно интересный. Если он что-то говорит, то говорит интересно и по делу, чем отличается от многих американских технических преподавателей, которые любят за ради дополнительных оплачиваемых часов нести какую-нибудь пургу. Вдобавок, у него всё в порядке с чувством юмора. Да, шутит он не часто, но в отличие от эстрадных юмористов, шутки у него очень добрые...

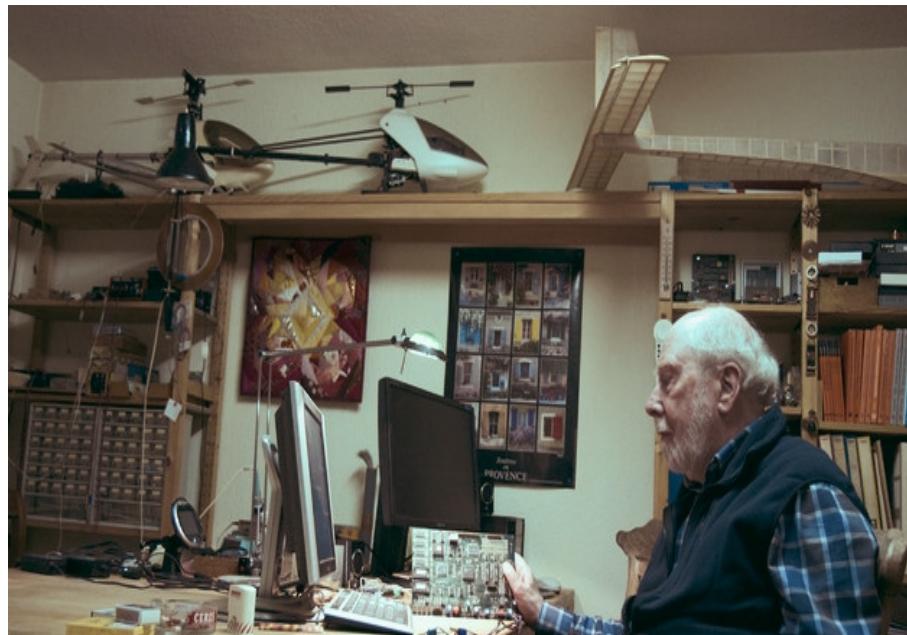


Рисунок 21. Вирт на пенсии не скучает...

3. История после истории. Наследники Паскаля

Ответвление – UCSD Pascal

После того, как Виртом в 1973 году было опубликовано обновлённое описание языка, Паскаль начал победное шествие по планете. Через год появилось уже 5 версий Паскаль для разных машин и к 1979-ому их было больше 80. В основном Паскаль использовался в европейских университетах и научных центрах. В США Паскаль приживался неохотно. Первым университетом, где Паскаль стал серьёзным проектом, был Калифорнийский университет в Сан-Диего. Проект был начат ориентировочно в 1974 году, точную дату сейчас определить затруднительно, т.к. у них довольно долго шли консультации с Виртом. Там стали разрабатывать компилятор на основе виртовской виртуальной машины и компилятора Pascal-P2.

Руководитель проекта Кеннет Боулз считал, что количество новых платформ будет постоянно расти, поэтому оптимальным вариантом будет некая прослойка между ОС и компилятором, чтобы компилятор работал только с небольшим количеством обобщённых объектов, а уж виртуальная машина транслировала бы эти объекты в конкретные железячные. Паскаль подходил просто идеально: во-первых язык сам по себе несложный, выучивался легко, а во-вторых, присутствовала виртуальная машина. По всей видимости Лисп, у которого тоже была виртуальная машина, показался ему сложноватым. Правда тут есть и ещё один нюанс. Существовавшая в то время Лисп-машина, которую в 1973-1974 году изготовили в Массачусетском технологическом институте, была заточена под вполне определённое железо, в то время как виртовская машина, как и язык, была без каких-либо «железных» привязок.

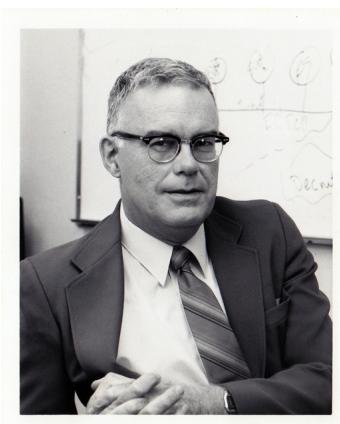


Рисунок 22. Профессор Кеннет Боулз задумался над созданием UCSD Pascal

Над названием размышляли не очень долго. Получилось простенько, но со вкусом — UCSD Pascal по имени самого уивера. Этот проект является основой для всех американских Паскалей и одновременно первым проектом для персональных компьютеров. Версия, которая получила широкое распространение, стала 1.3 в 1977 году. В 1978-1979 годах Марком Алленом и Ричардом Гливсом, которые учились тогда в UCSD, была сделана версия для компьютера Apple II на процессоре 6502. В дальнейшем этот вариант перекочевал на компьютер Apple Lisa обзаведясь графическим модулем.

USCD Pascal продержался до 1985 года и заглох из-за лучше прорекламированного TurboPascal'я.

Из интересного. В UCSD впервые стала применяться строка длиною в 256 символов. Введён новый тип — двоично-десятичное целое. Стали использоваться дополнительные служебные модули для наращивания функциональности. По умолчанию стоял явный запрет на использование GOTO, который можно было отменить специальной опцией компилятора. В UCSD Pascal впервые компилятор стал использоваться в связке с текстовым редактором, т.е. это была первая IDE.



```
>Edit: Insert Delete Copy Change Find Place Jump Adjust Set Margin Wrap Quit  
program Hello;  
begin  
    writeln ('Hello, world!')  
end.■
```

Рисунок 23. IDE для UCSD Pascal

Кроме виртуальной машины, на UCSD Pascal была разработана операционная система, однако судьба её неизвестна.

Почитать оригинальные мануалы по UCSD Pascal можно здесь [17].

Ответвление — «Адское» программирование американской военщины 😊

Наверняка все знают историю возникновения языка программирования АДА, поэтому опишу тут её всего парой слов. 😊

Аду паскалеводобным языком можно назвать лишь весьма условно, поскольку она унаследовала структуру и принципы работы от другого виртовского языка — Модула. А как самостоятельный язык Ада родилась в результате очень мощных телодвижений государственной машины. В 60-ых годах, когда автоматизация американской военной техники стала расти прямо как на дрожжах, языки программирования стали применять не только для научных вычислений всяких траекторий и прочих атомных взрывов, но и для управления в реальном времени различными устройствами. К началу 70-ых величина используемых программ управления превысило все разумные пределы как по объёму, так и по своим вариациям. Причём использовались как ассемблеры, так и высокоуровневые языки, которые программисты изучали или в колледже, или в университете. Порою чтобы модифицировать систему приходилось подолгу искать программиста, который бы разбирался в конкретном контроллере, его ассемблере, а то и вообще в некой не пойми откуда взятой модификации Фортрана, которая соответствовала стандарту настолько отдалённо, что никто его понять не мог, даже после двух бутылок Текилы. Назрела необходимость переходить на язык высокого уровня, который был бы прост и понятен любому не особо математически одарённому служителю Марса. В 1975 году, после предварительных согласований концепции языка, был объявлен всемирный конкурс на язык для встраиваемых систем. Это пришлось делать, поскольку ни один из уже существовавших в то время языков этим концепциям не удовлетворял. Что хотело МО США от языка высокого уровня, можно почитать здесь [18].



Рисунок 24. Программист, решившийся в начале 70-ых поработать на Пентагон

Заявок поступило очень много, потому что Пентагон, натерпевшись программистских мук выше крыши, премию выделил очень и очень соблазнительную. Предварительный отбор прошли только четыре, причём все четыре были основаны ни на чём ином, как на языке Паскаль. Язык, отвечающий всем установленным требованиям, был готов к 1983 году и получил название АДА. Далее Пентагон на удивление быстро продавил появление международного стандарта для этого языка. Видимо потому, чтобы с новым языком не произошла та же печальная история, которая в своё время случилась с суперпопулярным Фортраном. Там чуть ли не каждый уважающий себя программист считал своим святым долгом внести в Фортран усовершенствования без которых, как полагал программист, обойтись никак нельзя. Из-за этого до конца 60-ых по Америке ходило две сотни версий Фортрана, хоть и очень похожих, но мало совместимых между собой.

Ада, хотя и сохранила сходство с Паскалем, тем не менее по нашим современным понятиям больше напоминает язык Java. Тут нет никакого секрета. Java в то время даже в проекте не существовала, однако у Вирта уже имелся готовый язык MODULA, откуда адские разработчики многое чего тиснули в Аду.

Кардинальное отличие от старого Паскаля — модульность, сведение к минимуму возможности неконтролируемого взаимодействия объектов разного типа (сюда же относится и автоматические преобразования типов данных), зачатки параллельного программирования. Синтаксис стал более тяжеловесным, зато подавляющее число ошибок, которые у Паскаля могли быть выявлены только на этапе запуска программ, в Аде стал выявлять сам компилятор.

Из интересного. Системой на языке Ада управляется одна из веток Парижского метрополитена. Пентагон в 1987 году разрешил мировой общественности использовать язык, однако необходимо было пройти огромное число тестов, которые выявляли, даёт ли новый адский транслятор использовать стандартные адские возможности. На Аде написано бортовое ПО для самолётов Boeing-787 и военного транспортника C-130J «SuperHercules». В СССР во второй половине 80-ых был создан специальный комитет по изучению Ады, составились несколько учебных курсов и даже всесоюзный стандарт для языка («ГОСТ 27831-88», почитать можно здесь [19]). Было даже выпущено несколько книжек, в основном переводных. Из реального применения можно назвать только систему управления для Бе-200, которую делали для последних модификаций, совместно с американцами, планировавшими выпускать у себя такой же самолёт. Планировалась разработка системы управления полётами, но сделали её или нет, мне неизвестно. Перед прохождением нового самолёта Ил-86 международной сертификации, часть бортового ПО у него была переписана на Аде. В общем Ада в СССР и позже в России не оставила сколько-нибудь значительных следов, в отличие от Паскаля.



Рисунок 25. Книги изданные в СССР по языку Ада

Во второй половине 80-ых со всей очевидностью стало понятно, что идея «Ада — один язык для всего на свете» — это волюнтаризм и утопия. Образовался комитет по усовершенствованию стандарта, в котором повторилась история комитета Алгола. Главный разработчик первой Ады, Жан Ичбия, вынужден был покинуть комитет, поскольку категорически не был согласен с теми кардинальными изменениями, которые привели к усложнению реализации языка.

В новый стандарт (Ада-95) ввели:

- Зачатки ООП. Пошли по пути Модулы, т.е. фактически классом служил отдельный пакет (модуль) Ады, в котором описывались видимые и невидимые снаружи типы данных, а так же видимые и невидимые снаружи обработки методы этих типов. Получилось сложно даже по сравнению с Модулой;
- Интерфейсы взаимодействия с объектным кодом, написанным на других языках. В частности Фортран, Кобол, Си. Увы, в скорости Ада продолжала проигрывать Фортрану. С Коболом всё понятно, потому что самое большое и самое боеспособное подразделение Пентагона — это бухгалтерия с экономистами. Это единственное подразделение которое ни разу не проиграло ни одного сражения. ☺ Обмен осуществлялся через ключевое слово «pragma», к которому добавляется либо «Import», если надо что-то забрать из чужой программы, либо «Export» если надо что-то передать.
- Иерархические библиотеки. Это по следам ООП... ☺
- Распределённое программирование.

и многое чего ещё... Фактически этот стандарт является основой сегодняшней версии Ады.

На сегодняшний день Ада, хоть и считается языком сложным (а в интернетах про него даже пишут «мёртвый язык» 😊), продолжает успешно развиваться. Ада — составная часть комплекта компиляторов GNU (GCC). Там он называется GNAT. Нынешняя версия GNAT сделана на основе стандарта Ада-2012. Что интересно, этот проект разрабатывался по заказу BBC США изначально как OpenSource для того, чтобы для военных разработок можно было бы использовать аутсорсинг для удешевления разработки и сопровождения. Ну и чтобы самим можно было отщипнуть крошечку на пропитание детишкам малых, без опасения попасться на глаза военной прокуратуре. 😊

GNAT состоит из двух частей — коммерческой (версия PRO) и свободной (версия Community Edition). Так что в GNU GCC стоит свободная часть, но не вся целиком. Если зайти на сайт GNAT [20], то там можно скачать свободную часть вместе со средой разработки (которая напоминает ECLIPSE) в разделе Community -> Download. Есть версии для Windows и Linux как 32- так и 64- ёхразрядная. Есть для RISC и ARM процессоров. Если у кого-то завался RaspberryPi, для него тоже есть. 😊 GNAT обновляется каждый год. У GCC GNAT обновляется только вместе с остальным комплектом. Ещё одна фишка GCC — среда программирования там идёт отдельно и, почему-то, старой версии. Так что если хотите получить графическую IDE, то фактически приходится ставить две версии GNAT'а — текущую и предыдущую.

Теперь посмотрим, что нам может предложить GNAT for Windows [21]. Установочный экзешник весит почти 400 МБ. Это понятно, т.к. внутри должна быть какая-то квазисреда, типа MinGW или CygWin. Установка на Windows 7 идёт долго, но без проблем. Для примера накалякаем какое-нибудь консольное приложение:

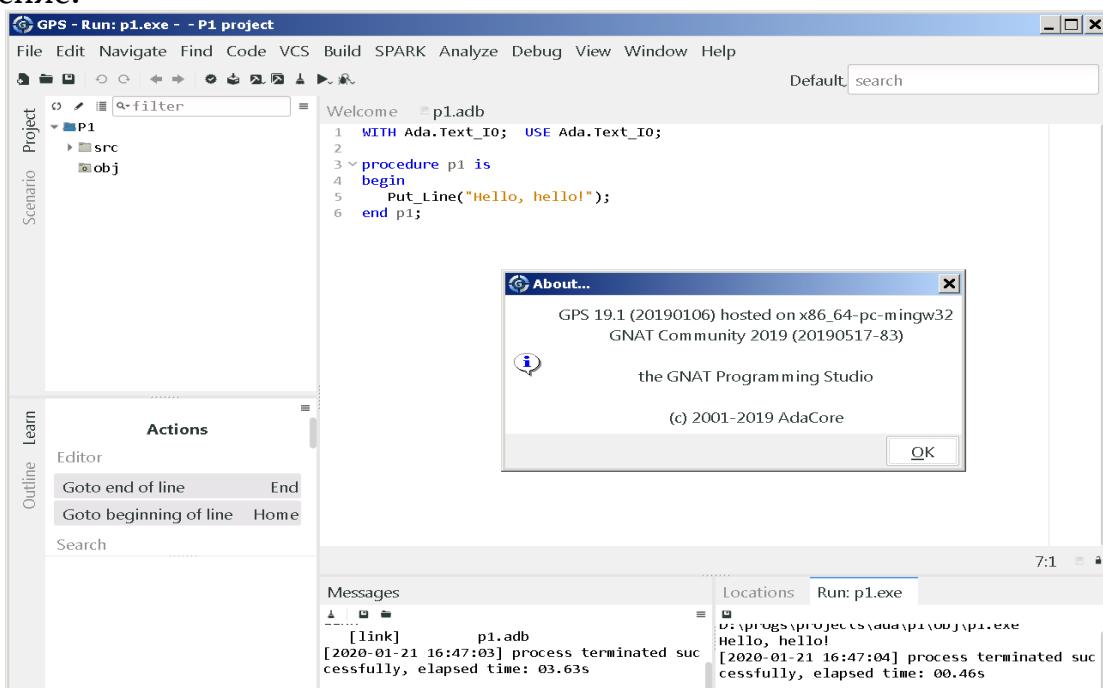


Рисунок 26. GNAT Programming Studio (GNAT PS) 2019

Как видно, у нас теперь стоит свежайший GNAT Programming Studio (GNAT PS) на основе MinGW. Сборка приложения в дополнение к исходникам добавляет нам ещё большую кучу файлов:

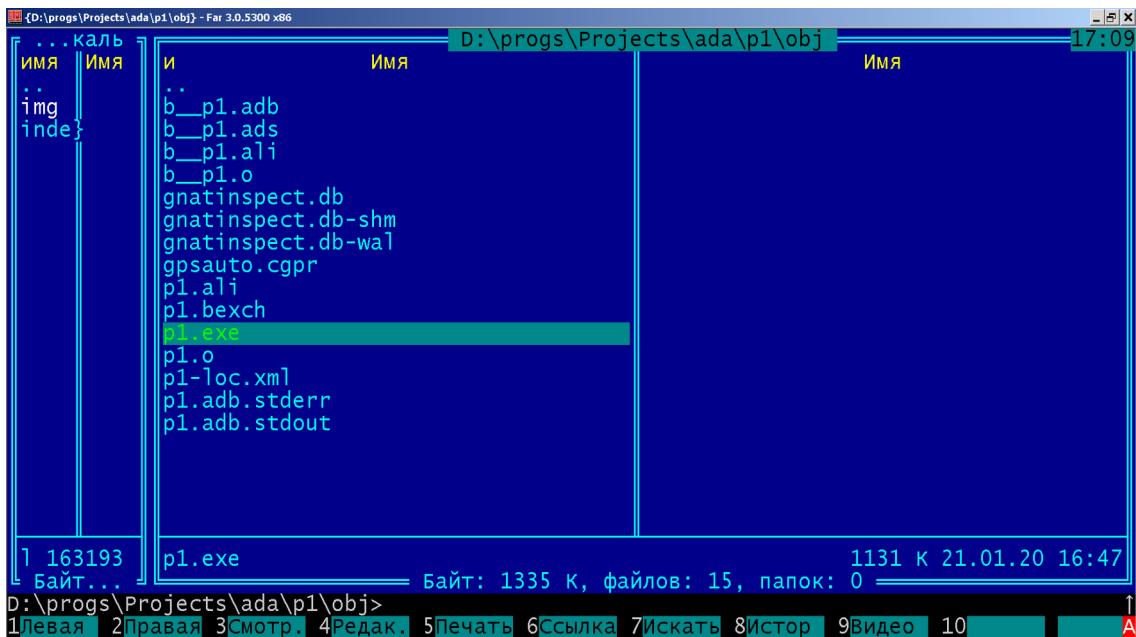


Рисунок 27. Программа на Аде после компиляции

Как видно экзешник весит солидно. FreePascal аналогичный программный текст компилирует с куда более скромным аппетитом. Впрочем, на выхлопе программы это не сказывается:

```
D:\progs\Projects\ada\p1\obj>p1.exe
Hello, hello!

D:\progs\Projects\ada\p1\obj>_
```

A screenshot of a terminal window. It shows the command 'D:\progs\Projects\ada\p1\obj>p1.exe' being run, followed by the output 'Hello, hello!'. Below the terminal window, a menu bar is visible with items 1Левая, 2Правая, 3Смотр., 4Редак., and 5Печать.

Рисунок 28. Работа программы написанной на Аде

Строка «WITH Ada.Text_IO» — это аналог паскалевского «Uses», т.е. подключение дополнительного модуля. В данном случае это модуль текстового ввода-вывода. Следующая, «USE Ada.Text_IO» нужна для того, чтобы использовать сокращённый синтаксис написания процедур\функций из этого модуля. Иначе текст программы выглядел бы так:

```

WITH Ada.Text_IO;

procedure p1 is
begin
  Ada.Text_IO.Put_Line("Hello, hello!");
end p1;

```

У Ады нет универсального ввода\вывода, как у Паскаля. Для каждого стандартного типа данных (текст, целые числа, вещественные числа) сделан свой отдельный модуль-пакет. Однако есть возможность как в C# на ходу преобразовывать значения того или иного типа непосредственно в строку. Можно заметить, что «end» процедуры снабжена ещё и именем процедуры. В Аде это обязательное свойство, как и в новом Фортране, Модуле или Обероне. Для программиста это очень полезно, чтобы он видел, к чему конкретно относится тот или иной «end».

Наверное всем будет интересно, как Ада общается с функциями операционной системы. Это похоже на Паскаль, только используется дополнительное ключевое слово «pragma». Вот, для примера, вызов системной функции «System()» чтобы запустить какую-нибудь чужую программу:

```

with Ada.Text_IO;
with Interfaces.C; use Interfaces.C;

procedure Call_Uname is
  function System (Arg : Char_Array) return Integer;
  pragma Import (C, System, "system");

  Retval : Integer;
begin
  Retval := System (To_C ("uname -a"));
  if Retval /= 0 then
    Ada.Text_IO.Put_Line ("Exit code:" & (Integer'Image (Retval)));
  end if;
end Call_Uname;

```

Как видно, сначала пишется заголовок функции «System» как он будет выглядеть в Аде, потом ключевое слово «pragma» и название «Import». Дальше в скобках идут опции для компилятора-линковщика:

- **C** — название языка, который делал импортируемую функцию;
- **System** — название этой функции в коде вашей программы;
- **"system"** — название функции в оригинале, т.е. в библиотеке, откуда она берётся.

Тут вы конечно же спросите — где же название самой библиотеки? А нету... ☺ Точнее говоря, если речь идёт о системной функции, то название библиотеки прикладывать необязательно. Для языков Си, Фортран и Кобол компилятор Ады и так знает название их стандартных ран-тайм библиотек. Это

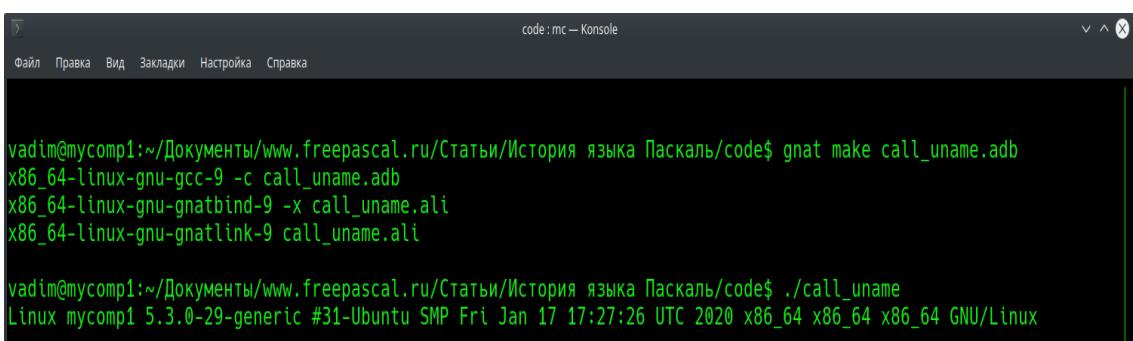
когда-то было очень удобно для взаимодействия с Фортраном, поскольку тот буквально напичкан по самые уши математическими функциями. А вот если речь будет идти, к примеру, о работе с сервером БД, то тогда вводится четвёртый параметр прагмы импорта — опции линковщику, куда название линкуемой библиотеки и вписывается.

Внутри вызова функции «System» можно заметить другую функцию — «To_C()». Она преобразует строку Ады в сишину строку (массив символов с завершающим нулём), иначе библиотечная функция не поймёт, что за параметр вы ей тут подсовываете. Есть и обратная функция — «To_Ada()», которая в случае необходимости преобразует сишину строку в ту, что используется в Аде.

Чуть ниже можно увидеть пример преобразования в текст «на ходу»:

Integer'Image (Retval)

Здесь у типа Integer берётся изображение (Image) из переменной Retval. На рисунке ниже приведён выхлоп программы вызывающей системную функцию:



The screenshot shows a terminal window titled "code : mc - Konsole". The window contains the following text:

```
vadim@mycomp1:~/Документы/www.freepascal.ru/Статьи/История языка Паскаль/code$ gnat make call_uname.adb
x86_64-linux-gnu-gcc-9 -c call_uname.adb
x86_64-linux-gnu-gnatbind-9 -x call_uname.ali
x86_64-linux-gnu-gnatlink-9 call_uname.ali

vadim@mycomp1:~/Документы/www.freepascal.ru/Статьи/История языка Паскаль/code$ ./call_uname
Linux mycomp1 5.3.0-29-generic #31-Ubuntu SMP Fri Jan 17 17:27:26 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
```

Рисунок 29. Вызов сторонней программы с помощью импортированной функции

Перспективы языка. Несмотря на то, что про него регулярно пишут как про забытый язык, он продолжает активно развиваться. В 2012 году вышла очередная версия стандарта, куда ввели термины, которые без бутылки и не разберёшь. ☺ Например, «контрактное программирование». Оказывается что это система предварительной и последующей проверки условий выполнения процедур. Т.е. прежде чем процедура запустится, выполняется некий код проверки, а стоит ли её запускать. После неё так же проверяются последствия запуска. Обычно такие вещи встраивают либо в сам код процедуры, либо в то место, где процедура запускается. В новой Аде теперь это отдельная программная конструкция. Правда эта штука касается тестов готовой программы, т.е. выполнение-невыполнение можно проверить только запустив готовую программу.

Честно признаемся, репутация Ады была изначально подмочена тем, что, во-первых, он специальный язык министерства нападения США, а во-вторых,

позиционировался как язык для встроенных систем. И в результате у него довольно долго средства ввода\вывода ограничивались только консолью и было тут с библиотеками по работе с БД. Впрочем, такая картина была у большинства языков во время создания Ады. Сейчас (имея в виду GNAT) — это полноценный универсальный язык программирования. У GNAT for Windows есть дополнительный установочный пакет по работе с GTK (куда, естественно, включён и сам GTK) и можно довольно спокойно создавать оконные приложения. Интеграция с динамическими библиотеками Си у Ады довольно спокойная, в отличие от, к примеру, языка Фортран, поэтому добавление интерфейса работы с той или иной системной библиотекой не представляет каких-либо трудностей. Так что если вас не пугает, что Ада уже далеко отошёл от Паскаля, его без проблем можно использовать для написания программ общего назначения.

Интерес к Ада может проиллюстрировать страничка в Community [22], где собраны учебные заведения со всего мира, которые в той или иной степени изучают или используют этот язык. Как не трудно догадаться, больше всего таких заведений в США. ☺

На русском языке про Аду можно почитать здесь [23]. Интересного там много, есть только одна проблема — контент сайта давно уже никто не пополняет. Активен только форум и если у вас с английским не очень, задавать свои вопросы можно на форуме этого сайта. Советские книги про Аду следует читать с большой осторожностью, т.к. они все про самую первую, старую Аду. Иногда приведённый код может быть не рабочим, так что к этому делу нужно подходить творчески. Для современной Ады есть Вики [24] на английском языке. Там очень много примеров, описания довольно краткие, так что уверен, проблем с пониманием не возникнет.

Читать стандарт Ады или системное описание языка — мучение неимоверное, занятие для настоящего мазохиста. ☺ По сравнению с книжками Вирта — это небо и земля.

Насколько целесообразно изучать Аду как язык программирования? Если это предписано вашей трудовой деятельностью — то вне всякого сомнения. ☺ С освоением синтаксиса трудностей никаких не будет, поскольку он похож на Паскаль. Однако тут же кроется источник постоянного раздражения на первых порах. Проблема в том, что подсознательно вы уверены, будто это тоже Паскаль. Увы — нет. Необходимо будет делать над собой усилие, чтобы постоянно искать — а как же называются те или иные элементарные функции в Аде, которые в Паскале вы знали назубок. И в каких модулях они лежат... Но успехи будут проявляться намного быстрее, чем при освоении какого-то принципиально иного языка, например, Лиспа... ☺. Если бы мне нужно было отказаться от Паскаля и в качестве альтернативы выбирать — Ада или Модула (который концептуально в той же плоскости, что и Ада), я бы однозначно выбрал Модулу. К Паскалю она намного ближе, а вот Ада сам по себе язык на порядок сложнее и Модулы и даже Паскаля.

Тем не менее, если браться за язык серьёзно — это хорошая штука. Сопутствующих, т.е. идущих в стандартном комплекте, модулей у неё довольно много, сравнимо с Delphi или FreePascal. Проблем при решении той или иной задачи с использованием Ады я не вижу. Ну, естественно, речь не идёт об этапе первоначального знакомства... ☺

1-ое поколение наследников — Apple Pascal, Borland TurboPascal и MS QuickPascal

Самая сложная часть, потому что все всё об этом знают... ☺ С этих экземпляров начинается кардинальный отход от одной из виртовских концепций Паскаля — универсального языка. Паскаль начинает проектироваться под один-единственный процессор. Все потомки первого поколения развивались параллельно Аде, но в отличие от Ады использовали предыдущую виртовскую платформу, поскольку Виртовские нововведения показались им очень уж революционными. Не для бизнеса, однако... ☺

Apple Pascal. Он появился первым, поскольку для Apple уже был разработан UCSD Pascal. В самой фирме очень ретиво ухватились за такой подарок судьбы и использовали на 100%. В 1979-1980 годах виртуальная машина UCSD p-System была творчески переработана в настоящую операционную систему. Естественно эта ОС была написана именно на Паскале. Несколько версий компьютеров Apple выходили с этой ОС, однако бешеной популярностью они не пользовались. Во-первых это была ещё пока ОС с чисто текстовой консолью, а во-вторых, и в главных, профессиональных программ для неё было довольно мало. Зато Паскаль, как язык программирования, входил туда составной частью. По отзывам современников эта ОС отличалась очень высокой стабильностью и завалить её было практически невозможно. Всего вышло 4 версии ОС.

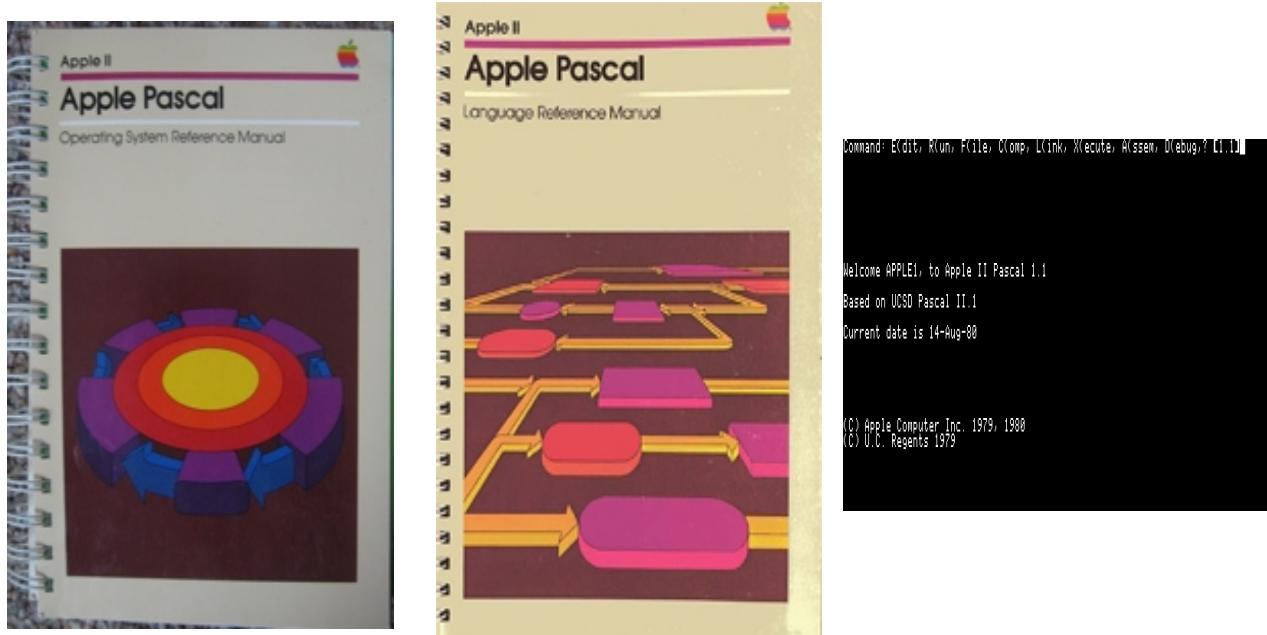


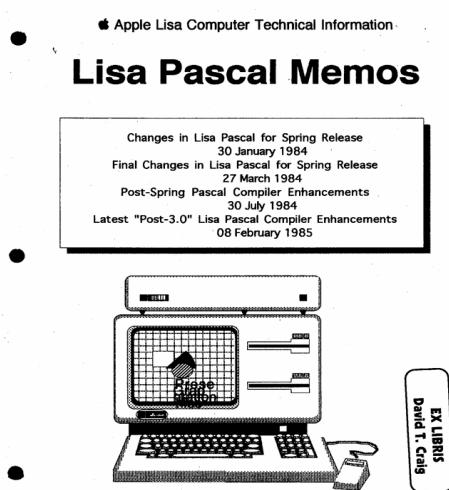
Рисунок 30. Apple Pascal — язык и ОС

Затем появился микрокомпьютер Apple Lisa, которые имел выдающиеся

по тем временам характеристики: 32-битный процессор, ОЗУ 1024 кБ, встроенные графическая и звуковая карты. По наследству к Лизе перешёл и Паскаль, однако внешность его облагородилась — редактор стал похож на те, что мы пользуемся сегодня и вид его стал псевдографический. Т.е. он был конечно графическим, но оперировал только текстом. Для графической подсистемы был разработан новый паскалевкий юнит Graph.



Рисунок 31. Микрокомпьютер Apple Lisa с Apple Pascal на борту



TurboPascal. С ним я впервые познакомился в прошлом веке, в 1996 году. Дело было так: однажды моя коллега, в высшей степени милая и симпатичная девушка, задала вопрос — умею ли я программировать? А в то время моя должность была связана исключительно с электроникой. Как настоящий мужчина я немедленно принял очень мужественную позу, которую, естественно, почерпнул из американских боевиков, и спросил — а с чего это вдруг, детка, возник такой странный и немного интимный вопрос? ☺

Как выяснилось — ничего противоправного не замышлялось. «Детка», оказывается, училась в институте, вот-вот должна была начаться сессия, а программа хоть и была написана но, отчего-то, совершенно непонятно почему, не работала. Ну какой-же настоящий мужчина отказывает юным и прелестным девам в таких пустяках? ☺ Вот и я говорю — какой вопрос, детка, конечно умею. Неси её сюда и за пять минут проблема будет решена. Вы может быть подумали, что, как и всякий настоящий мужчина, я обманул бедную наивную девушку? Нельзя сказать чтобы это было полностью так, потому что кое-какой опыт программирования у меня имелся, например на калькуляторах типа МК-45 и МК-61, плюс ещё немного Бейсика на ДВК-3М... ☺

Когда был принесён текст программы, немедленно выяснилось, что быть настоящим мужчиной не так-то просто — код оказалась на Паскале, о существовании которого я узнал только в эту секунду. ☺ Не желая ронять своё мужское достоинство в глазах девушки, у которой глаза светились надеждой, но уже с какой-то долей обречённости, я принял ещё более мужественный вид (хотя, казалось бы, куда же ещё... ☺), поглядел на код сначала левым, а потом правым глазом, картинно сморщил лоб и сказал, что задача оказалась несколько сложнее, чем я себе это представлял. Надо некоторое время над ней подумать. ☺ Но сожри меня 4096 чертей и 16 каракатиц, если к завтрашнему утру я чегонибудь не придумаю! ☺

Девушка немедленно просияла лицом, сказала что я самое первое настоящее чудо в её нелёгкой жизни и как она рада, что знакома с настоящим рыцарем без страха и упрёка, пускай даже без плаща и кинжала. И ушла ждать завтрашнего утра, когда наступит её, принцессы, избавление от дракона-сессии... ☺

Первое что я сделал немного прия в себя — кинулся в ближайший книжный магазин. На моё счастье, там оказалась одна-единственная книжка по TurboPascal, причём в последнем экземпляре, которую я немедленно купил. Пойти в библиотеку мне отчего-то ума не хватило. Впрочем, настоящие мужчины туда ходят только в самом крайнем случае. ☺

4 часа, проведённые над книжкой и текстом программы показали, что проблема не очень то и серьёзная. TurboPascal оказался языком несложным и, вдобавок, в какой-то степени похожим на Бейсик. Осталась только одна проблема — дома на моём компьютере TurboPascal'я не было и установить его не представлялось возможным. Сразу хочу сказать, что задача была с использованием графики, поэтому проверить то, что я придумал, на Бейсике —

совершенно не реально. Утром, с видом несколько потрёпанным, но тем не менее победоносным, мы вместе с девушкой ввели поправки в её текст и оказалось, что ошибся я всего в двух местах, но алгоритм был верен. Так я впервые познакомился с языком программирования Паскаль и он стал моим любимым языком... ☺



Рисунок 33. Так я познакомился с TurboPascal... ☺

Как коммерчески успешный проект, TurboPascal появился исключительно благодаря бешено критике Паскаля сишиками. Получилась своеобразная реклама, причём за счёт сишиных компаний. ☺ Филипп Кан, работник, а затем и владелец компании «Borland», как и Вирт является выпускником Высшей Технической Школы в Цюрихе, поэтому про Паскаль знал очень много, а как математик, много в нём понимал. ☺ Сначала фирма «Borland» занималась разработкой ПО для ОС СР/М. Однако довольно быстро стало понятно, что СР/М ждёт забвение и надо бы заняться программированием для какой-нибудь более коммерчески успешной платформы. MS-DOS показалась подходящей и перспективной. Вот только что под неё писать? Занимая свою, хоть и большую, но довольно специфическую нишу, MS-DOS предназначалась не для большой и тяжёлой работы, как суперкомпьютеры. Но с другой стороны, возможности для такой работы следовало бы создать уже сейчас, не дожидаясь когда во вновь образовавшуюся рыночную нишу проскользнёт кто-то ещё, не менее хитроногий. В общем поступило предложение — создать программу для написания программ. Естественно, не жрущую довольно небогатые ресурсы, тем не менее, удобную для разработчика. С Бейсиком связываться не стоило —

засудят в пять минут. Фортран или там Лисп на тот момент пока были тяжеловаты для персоналок, хотя тот же UCSD на Паскале написал компилятор для Фортрана. Но здесь можно было нарваться уже на недовольство IBM, чьей неофициальной собственностью полагали Фортран. А судебно спорить с IBM — дело совершенно экономически неэффективное, хотя в рекламном смысле очень даже интересное. Оставался Си, который вроде как можно было делать самостоятельно, благо что спецификации были в широком доступе, как и у Паскаль. Правда Си оказался в спецификациях сильно сложноват, но тут уж выбирать особо не из чего. А вот спецификации на Паскаль были просты и понятны. Языком Си Borland тоже занялась, но значительно позже и основываясь на опыте коммерчески успешного Паскаля.



Рисунок 34. Филипп Кан после того как сообразил из чего можно извлечь бешеную прибыль
(из Паскаля) ☺

В общем то Паскаль сам прыгал в руки, тем более, что по нему к тому времени было с одной стороны много работ и достаточно практического опыта, а с другой стороны, в отличие от американских языков, он был более-менее открыт. «Borland» решил перелопатить UCSD Pascal, поскольку тот был как раз для персональных компьютеров, но в отличие от Apple лицензиями не закрывался. Однако взяли не непосредственно его, а выкупили один из модифицированных последователей у датского программиста Андерса Хейлсберга, дабы с финансовой точки зрения всё было чисто. Тот уже портировал Паскаль под MS-DOS, поэтому оставалось его слегка причесать, сделать красивым и удобным. Была создана текстовая IDE в стиле текстового редактора WordStar, который в Америке тогда пользовался бешеной

популярностью. Хейлсберг кстати тоже пошёл работать в Borland и даже участвовал в разработке Delphi. Однако в 1996 году его переманила высокими гонорарами Microsoft, для которой он сел писать язык C#.

TurboPascal 1.0, который вышел в 1983 году, мы сегодняшние вряд ли бы узнали: никаких окон, никаких мышек, управление — горячими клавишами. Но для тех времён дело вполне привычное и даже вполне передовое, судя по тому, что за первый месяц «Borland» набрал заказов аж на 150 000 долларов... ☺ Подобный невзрачный интерфейс продержался с версии 1.0 до 3.0 включительно.

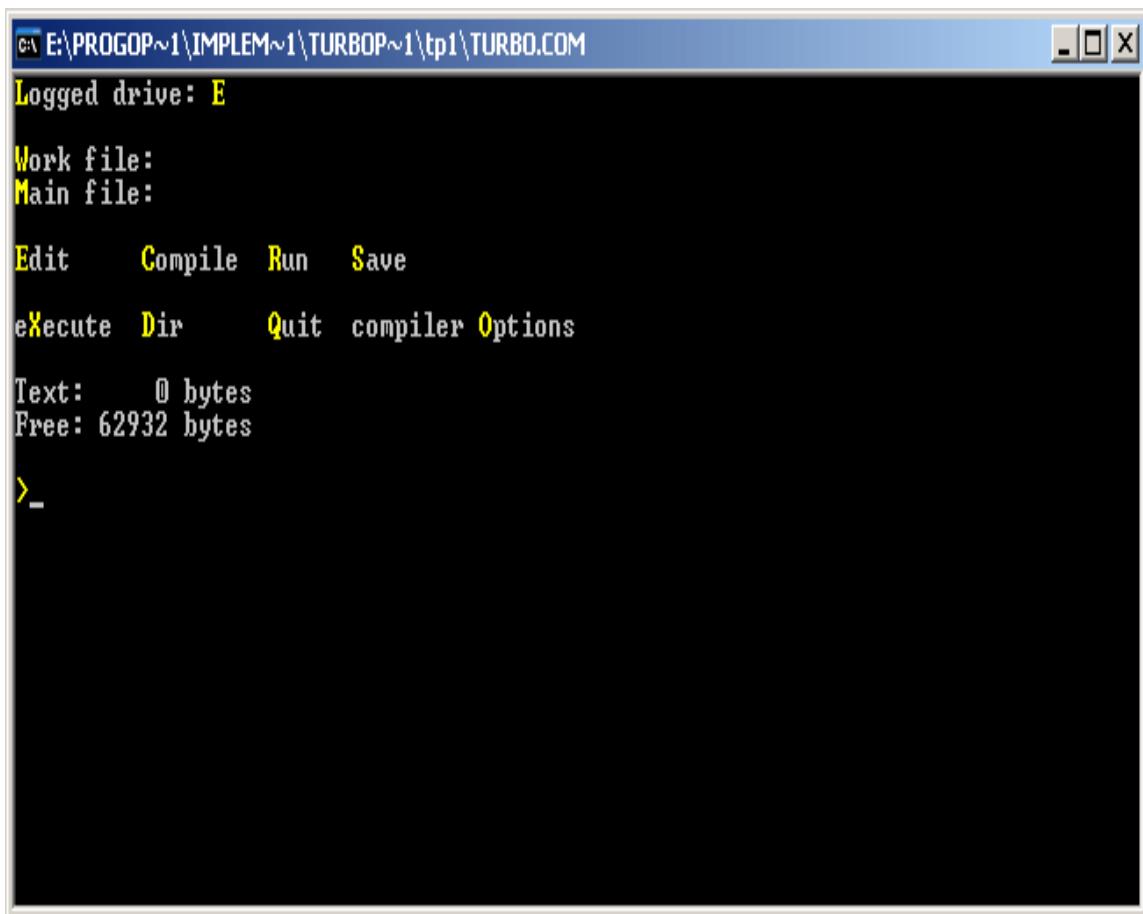


Рисунок 35. Turbo Pascal 1.0

Из вкусностей TurboPascal стоит упомянуть достаточно подробные и толстенькие книжки по языку и работе с этой средой и компилятором идущие в комплекте. Кстати говоря уже тогда с большим количеством примеров. Это было серьёзное преимущество, т.к. например, для языка Си других фирм, предполагалось покупка обучающих книжек самостоятельно, поскольку мануалы от производителей были очень уж тощенькими.

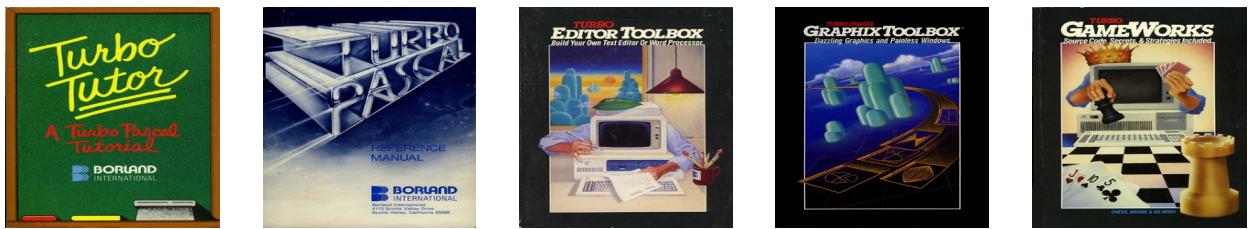


Рисунок 36. Руководства к разным версиям Turbo Pascal

В версии 4.0 уже можно узнатъ наш любимый многооконечный интерфейс, написанный без использования ООП, которого пока там ёщё нет.

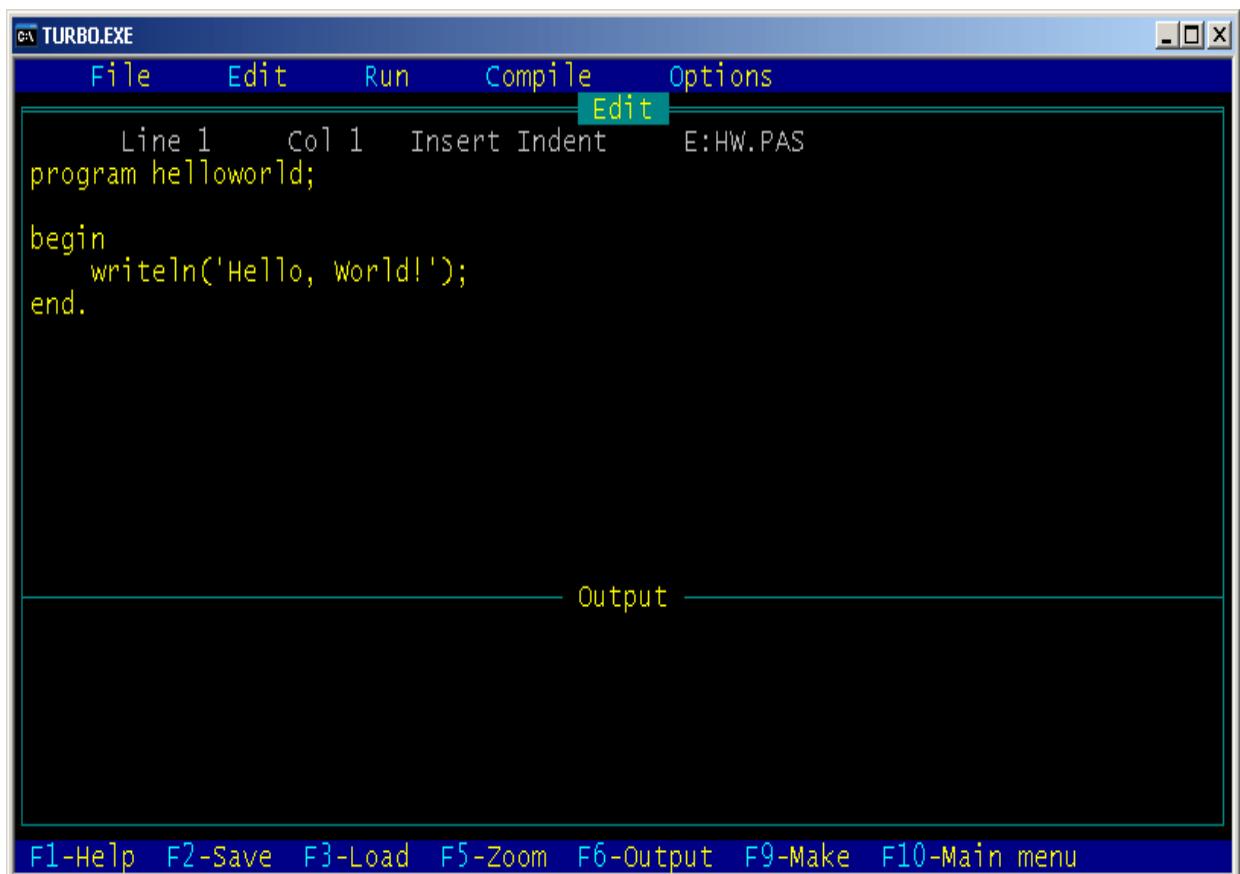


Рисунок 37. Turbo Pascal 4.0 с новым интерфейсом

Версия 5.5 обзавелась ООП, которую взяли из Apple Pascal. Эппловцы отдали ООП довольно легко по той простой причине, что уже разрабатывали ООП в виде классов, как это потом появилось в Delphi.

Версия 6.0 обзавелась интерфейсом, который хорошо помнят все советские, а так же российские студенты и школьники... ☺ Этот интерфейс сделан на основе ООП с использованием специальной библиотеки TurboVision.

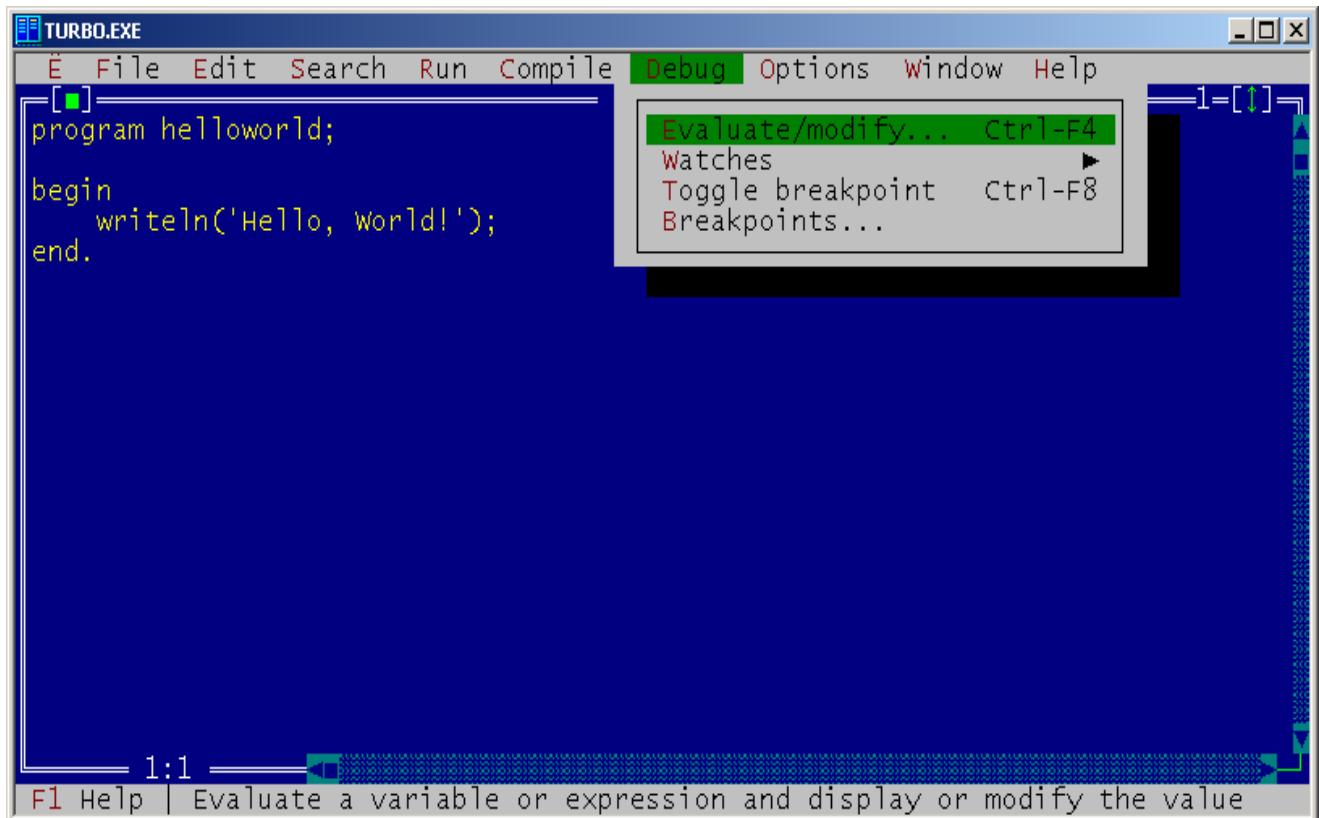


Рисунок 38. Turbo Pascal 6.0 с ООП-интерфейсом на основе TurboVision

Версия 7.0 откровенно говоря чувство новизны уже никакого не вызывала. Её могли бы пронумеровать 6.2, поскольку изменилась библиотека TurboVision на версию 2.0. Добавление виндовых заголовочных файлов желания программировать под виндовс вообще не вызывало по той простой причине, что кодировка русского языка изменилась по сравнению с той, что была в текстовом редакторе IDE. Даже выпуск чуть попозже Borland Pascal for Windows 1.0 дела не исправил — программировать оконные приложения стало очень сложно, причём нет никакой разницы, делаете это вы в Паскале или в родном окошечном Си.

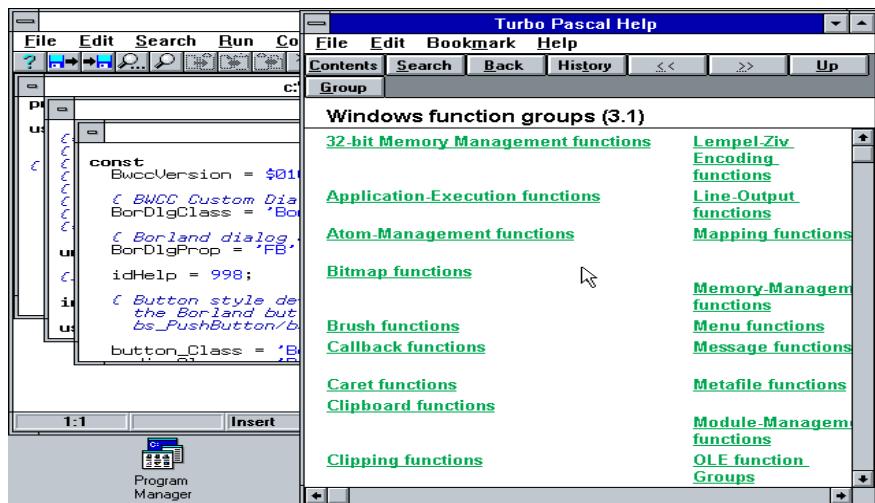


Рисунок 39. Turbo Pascal 1.5 for Windows

Впрочем, «Borland» тоже прекрасно понимала, что в виндовом программировании всё печально, поэтому специально для винды прилагалась отдельная программа редактор ресурсов — Resource Workshop, которая довольно сильно облегчала создание пользовательского интерфейса.

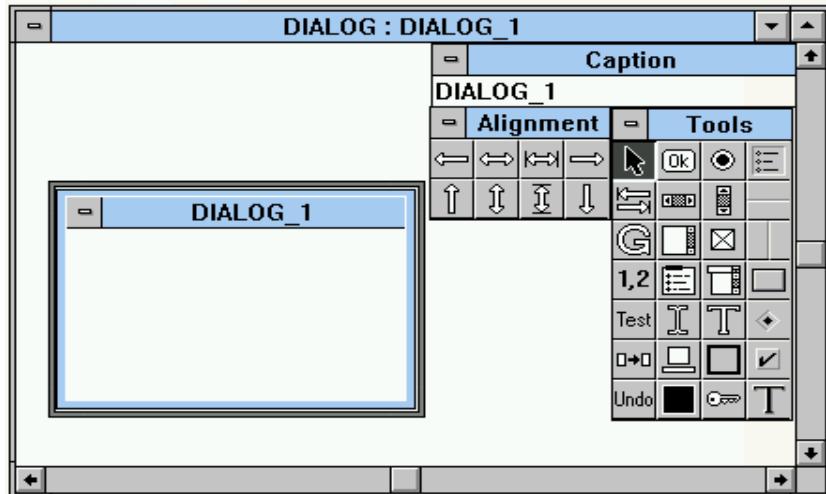


Рисунок 40. Resource Workshop — можно быстро создать пользовательский интерфейс под Windows

Совсем коротко о новшествах в разных версиях TurboPascal можно почитать в Вики [25].

Quick Pascal. А что же конкуренты? Конкуренты, естественно, не дремали. «Microsoft», видя какие бешеные прибыли приносит Борланду Паскаль, на время забыла о своём Бейсике и выпустила довольно неплохую версию QuickPascal. Причём, что интересно, не только для DOS, но и для Xenix. В комплекте шли обучающие книжки.

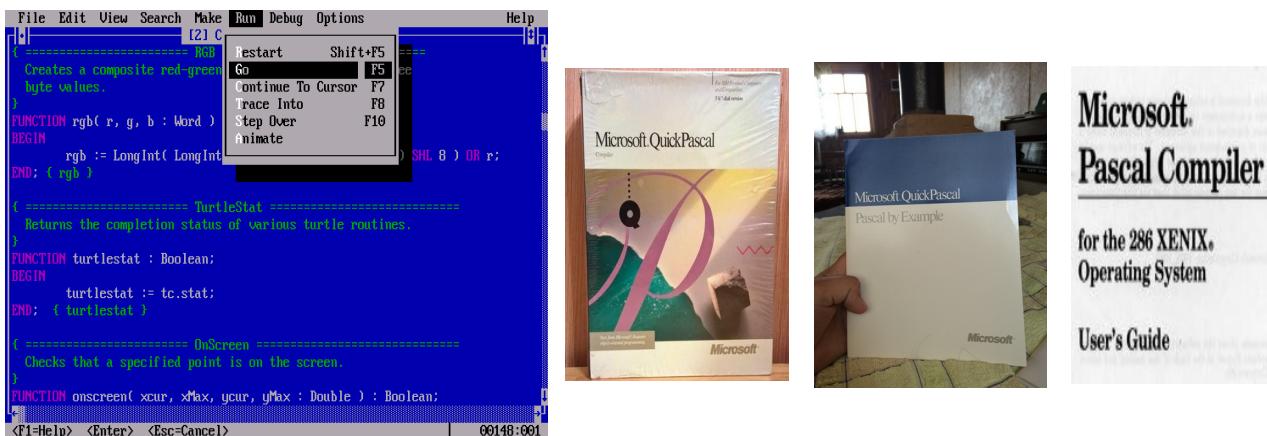


Рисунок 41. Microsoft Pascal

От себя хочу сказать, что QuickPascal по возможностям был примерно

аналогичен TurboPascal'ю. Однако была у него и ещё одна фишка, которая, если бы он продержался подольше, загнала бы TurboPascal в могилу. Дело в том, что Borland'овцы в попытке ускорить всё и вся перехетрили сами себя. Наверняка многие ещё помнят знаменитую «RunTime 200» у TurboPascal 7.0 на процессорах с частотой более 200 МГц. Borland даже тогда специальный патч выпускал. Так вот, QuickPascal благополучно продолжал работать и на процессорах с более высокой частотой, ни в какие ошибки не вываливаясь...

«Borland», чтобы ущучить конкурента, совершил героический маршбросок на его поле с продуктом TurboBasic.



Рисунок 42. TurboBasic от Borland

По всей видимости этот продукт я тоже в своё время попробовал, но, в отличие от QuickPascal, каких-либо впечатлений он у меня не оставил. Только помню, что был такой и всё. В общем, ничего интересного. У вершителей программного мира тоже никаких особых потрясений не было. До судебных разбирательств, как это принято в современной ИТ-индустрии США, дело тогда не дошло. Ходят упорные слухи, что «Borland» и «Microsoft» миром договорились не лезть в зоны влияния друг к другу, после чего TurboBasic и QuickPascal тихо исчезли сначала из продаж, а потом и вообще с рынка. Это явно свидетельствует о том, что слухи не лишены оснований... 😊

2-ое поколение — Delphi (ObjectPascal)

Если в первом поколении потомков, Apple ещё пыталась делать Паскаль для разных процессоров (правда тут следует уточнить — вызвано это было исключительно тем, что ихние компьютеры переходили с процессора на процессор), то вот Delphi стала первым Паскалем, который был железно прикован к одному единственному процессору, название которого я здесь писать не буду — боюсь обвинений в рекламе, а так же к одной единственной ОС... ☺

Ужасающая трудность программирования оконных приложений под винду вызвала резкий спад доходов у Borland. А поскольку в Америке быть неудачником считается намного хуже, чем быть даже нищим, компания, организовав мозговой штурм на тему «как перестать ломать мозг над программированием и начать программировать», придумала новую фишку — визуальное программирование. Нет, если подходить строго, то визуальное программирование было придумано ещё до Borland'a. Например, Microsoft использовало его ещё в программе FoxPro 2.0 for DOS. Это было не изобретение Microsoft, они FoxPro таким уже купили, но здравую идею поддержали. Аналогичный принцип они применили позже в своём Visual Basic for DOS. А вот лёгкость применения визуального программирования в винде все разглядели только в Delphi. Как показал опыт применения Resource Workshop в TurboPascal for Windows, простое использование ресурсов для пользовательского интерфейса программирование изображения интерфейса облегчает, однако остаётся ещё одна сложная часть — обработка событий этого интерфейса. И вот тут Borland выступила пионером. Правда добилась она упрощения обработки событий за счёт существенного увеличения как размеров готовой откомпилированной программы, так и за счёт занимаемой ею ОЗУ после запуска. Идея была в том, что помимо использования элементов интерфейса из ресурсов, каждая программа снабжалась готовым программным модулем обработчика стандартных событий для элементов интерфейса. Для простых случаев программирование превратилось в усладу души для художественно одарённых личностей — накидал в художественном разнообразии элементов на форму, внёс пару строчек кода в событие мыши OnClick — и программа готова. Рыба событийного управления программой уже находится в стандартном модуле, остаётся только назначить процедуры для тех или иных событий. А вот обработку этих событий с прикреплёнными процедурами выполняет рыба, программист сюда не только не вмешивается, но уже и не знает об этом.



Рисунок 43. Мы тоже умели программировать GUI 😊

Пошла новая волна критики уже Delphi по поводу того, что в нём программист быстро разучивается или вообще не учится программировать именно из-за простоты создания интерфейса. Типа, «накидал кнопок на форму и программа готова». Причём, что интересно, критика эта опять со стороны сишиков. 😊 И вот это опять заставляет подозревать исключительно коммерческую составляющую такой критики. Если к Delphi 1.0 относились ещё несколько настороженно, то вот 2.0 уже воспринималась на «ура». Тем более, что её подготовили, в отличие от 1.0, как строго 32-ухбитную, хотя и продолжала позиционироваться как система программирования для Windows 3.1.

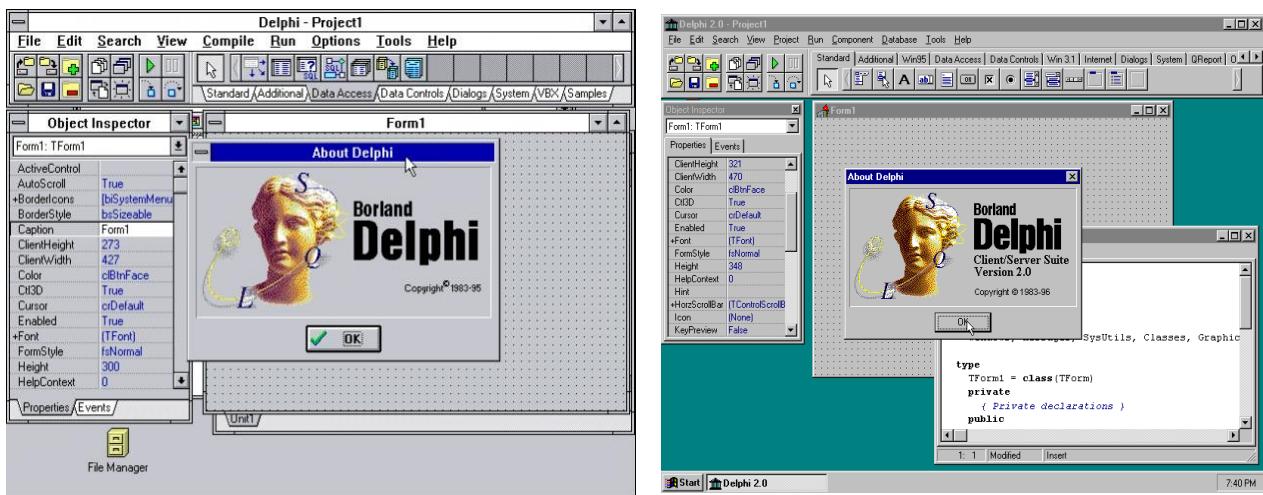


Рисунок 44. Delphi 1.0 и Delphi 2.0 пока ещё для Windows 3.1

Visual C++ от Microsoft в популярности потерял довольно значительно. Как раз в это время ими были предприняты усилия не только по выведению

своих продуктов на новый уровень, но и по ослаблению стана конкурентов. Вышла Visual Studio 4.0 (до этого Visual C++ и Visual Basic шли отдельными продуктами). Был перекуплен Borland'овский ведущий инженер-разработчик Андерс Хейлсберг. Его посадили заниматься языком C#, который, вот ведь чудо из чудес, стал напоминать Delphi... ☺

До версии 7.0 включительно Delphi шёл по эволюционному пути развития — основной набор компонентов был один и тот же, кое-что добавлялось по мере необходимости, например, работа в сети. Кое-какие компоненты менялись, например, простенький компонент для отчётов QuickReport сначала поменяли на более крутой ReportSmith, а потом на RaveReport, которые были самостоятельными программами. Вид среды разработки при этом практически не менялся. Революция произошла в Delphi 8, которая вдруг ни с того ни с сего в конце 2003 года резко перепрыгнула на платформу .NET. С одной стороны конечно понятно, что с Microsoft надо дружить, но вот зачем полностью отказываться от столь славной истории предыдущих Delphi — совершенно непонятно.

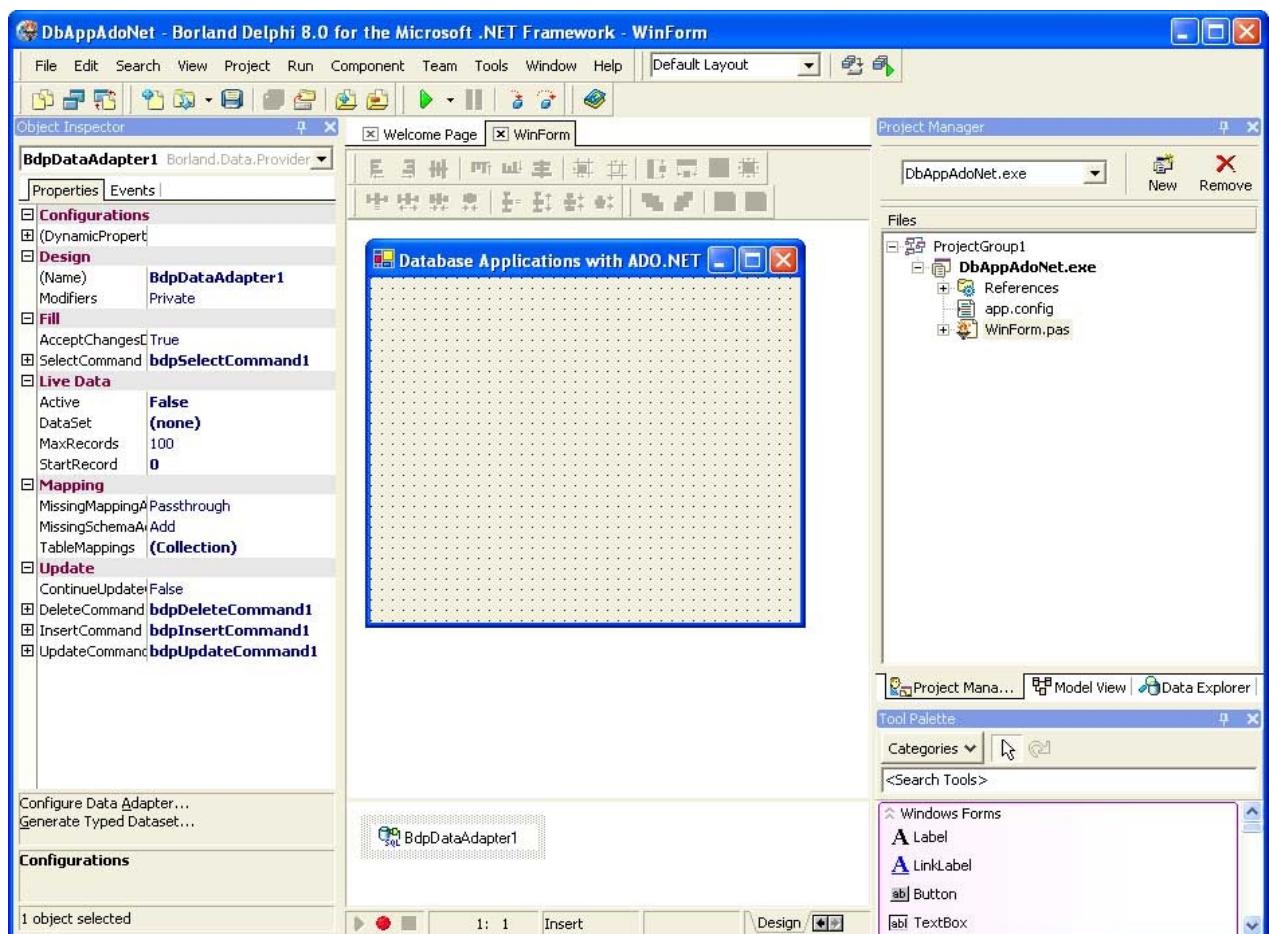


Рисунок 45. Borland Delphi 8 for .NET

Delphi как-то очень уж подозрительно стала напоминать новое Visual Studio, что и неудивительно, т.к. новая IDE со всеми своими элементами была

сделана на .NET. Ходили упорные слухи, что Microsoft переманило к себе команду разработчиков Delphi в полном составе и те, решив не терять бабки со старой работы, решили поработать и на наших и на ваших. ☺ Теперь стало сильно непонятно — если ты хочешь написать приложение для .NET зачем нужно Delphi, когда есть отличная штуковина по имени Visual Studio и где совершенно не нужно заморачиваться с расхождением в синтаксисе между C# и Delphi for .NET. С другой стороны, после перехода на 8-ую версию куда девать весь наработанный старый код? Популярность новой Delphi из-за этого резко двинулась к плинтусу. И мало того что новый, так и старый Delphi тоже стал резко терять свою популярность, потому что никто из разработчиков уже не мог предугадать, а что в дальнейшем ожидать от Delphi.

Надо сказать, что это не первая попытка Borland'a перебраться на другую платформу. Наверняка все ещё помнят проект Kylix для Linux. Была разработана кроссплатформенная библиотека CLX, которая и в Linux и в Windows работала на основе Qt. На мой взгляд провал Kylix случился из-за жёсткой привязки только к одной графической библиотеке Qt, что резко ограничивало разработчиков Borland в самостоятельной разработке. Тут вдобавок свою роль сыграла привязка Qt к объектно-ориентированному интерфейсу. Наверное для C++ в этом нет большой беды, т.к. он может напрямую читать Qt-шные библиотеки. А вот для других языков тут существует непреодолимое препятствие, т.к. у всех других языкам объекты используют другие принципы и построения и работы. С Gnome, наверное, было бы тоже самое, т.к. Kylix предполагалась кроссплатформенной и пришлось бы вместе с ней ставить дополнительные графические библиотеки, если вы собирались вести проекты не только в Linux, но и в Windows.

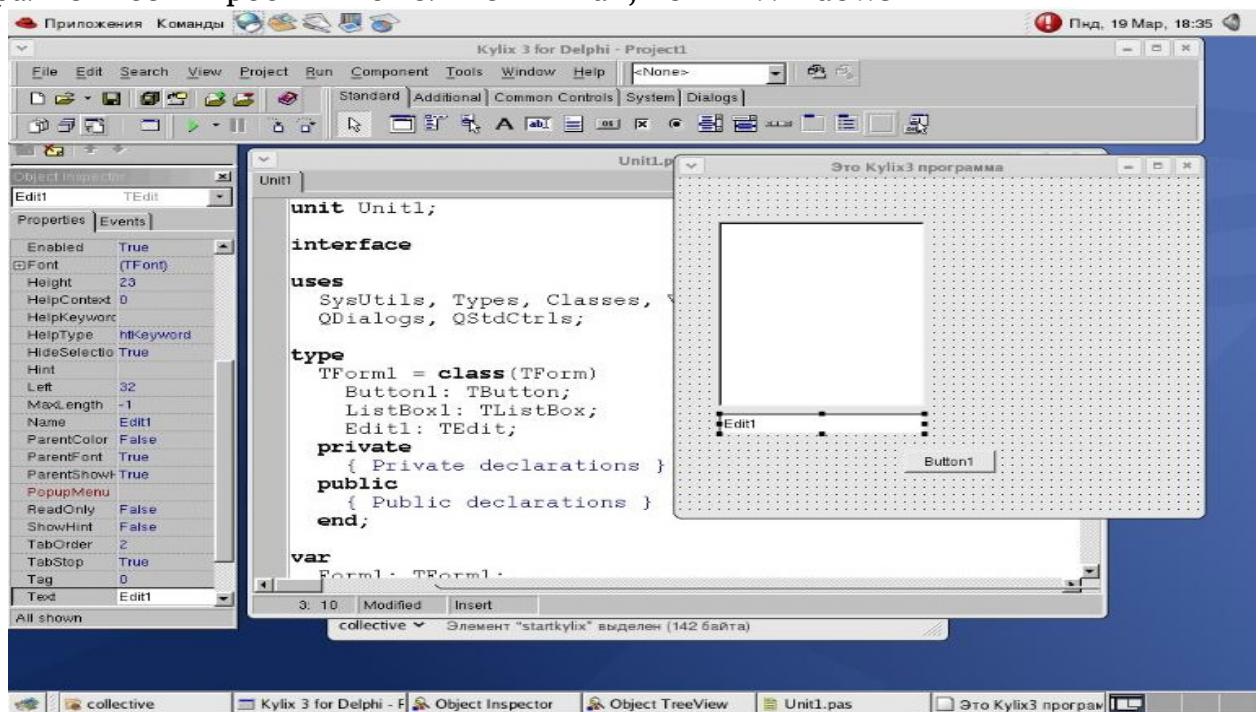


Рисунок 46. Borland Kylix для Linux

Поняв свою ошибку, Borland вернуло код Delphi 7 в следующую версию — 2005, правда оставив IDE работающее на .NET. Тогда же было принято решение отказаться от Linux — кроссплатформенная библиотека визуальных компонентов CLX была изъята из среды разработки.

В результате всех этих пертурбаций разработчики в своей любимой Delphi сильно разочаровались и несмотря на героические попытки преемников Borland исправить ситуацию, популярность Delphi так и не приблизилась к той, что была вначале. Даже добавление возможности делать проекты на C#, а потом и на PHP. Но возникает логичный вопрос — зачем для этого покупать какую-то левую IDE, если есть VisualStudio? Ну ладно, на PHP в VisualStudio писать нельзя, но зато в ней есть ASP, что гораздо удобнее — не надо держать в голове особенности нескольких языков. А вдобавок Microsoft тут ещё подложила довольно таки изящного поросёночка, в виде бесплатной версии VisualStudio, которая была менее крута чем платная, окошечные приложения на ней делать всё равно легко. У Delphi тоже появлялась бесплатная версия, но она была какая-то уж совсем скучоженная.

Кстати говоря, сама Borland, поняв свой полный провал, довольно долго пыталась избавится от подразделения, которое занималось связкой Delphi/C++. Сначала эта разработка была выделена в отдельное автономное подразделение CodeGear, чтобы своими долгами не портила общую финансовую статистику. Продать удалось только когда Delphi достигла версии 2010 и уже после полного перехода на Unicode (которое произошло в версии 2009, что довольно таки подозрительно, т.к. все новые Windows были ни на чём другом, как на Unicode). Увы, и новой компании вернуть былую популярность для Delphi так и не удалось...

3-е поколение — VirtualPascal, TMT Pascal, GNU Pascal, FreePascal

Может показаться странным, что такой интересный язык как Паскаль ограничился тем, что вы прочитали выше. Даже несмотря на то, что TurboPascal по тем временам оказался довольно дешев, должны быть какие-то ещё более интересные альтернативы (и в плане цены тоже... ☺). Действительно, альтернативы были. Например, для микрокомпьютера Amiga был разработан Паскаль, который шёл в дополнительном пакете ПО. Для микрокомпьютера Commodore 64 Паскаль был разработан не где-нибудь, а в самом Оксфорде и стоил всего 13 фунтов стерлингов. Даже Z-80, у которого была чудовищная по своей неудобности клавиатура, не обошла стороной паскальная эпопея. Однако, поскольку все эти компы чаще всего использовались как игровые, языки программирования на их основе как-то большой популярностью не пользовались, хотя разработчики тех же игр программировали вовсю (не только на Паскале ☺). Для самого популярного компьютера всех времён и народов, с альтернативными версиями Паскаля довольно долго было трудновато до тех пор, пока в конце 80-ых во всю ивановскую не развернулся Ричард Столлман — отец-основатель свободного (не бесплатного ☺) программного обеспечения.



Рисунок 47. Ричард Столлман — основатель проекта GNU и Free/OpenSource

Может возникнуть вопрос, а как же в такой в высшей степени закоммерциализированной стране, где даже во сне все жители спят и видят бабки, могла возникнуть такая очень странная организация свободного

программного обеспечения? Тут всё очень запутано. Сам Столлман всегда уточнял, что термин «FreeSoftware» (а до определённого времени это так и звалось) совсем не означает «бесплатно». Видимо поэтому в последствии все стали звать её «OpenSource», что намного ближе к столлмановской идеи. Так почему же?

В 80-ых годах, когда Столлман и создал организацию, стало совершенно понятно, что дела в стране катятся в то место человеческого (и не только ☺) организма, которое в приличном обществе предпочитают не упоминать. Сытые 60-ые и 70-ые похоже канули в небытие. Казалось бы ничего страшного, но отчего-то большие компании перестали расширяться, открывать новые филиалы. А чуть позже стали закрываться производства автомобилей, которые в Америке были всегда на первом месте. Явно что-то пошло не так и уже надо бы думать о 30-ых, хотя и не хотелось. В отличие от 30-ых, государство ясно давало понять, что благотворительностью больше заниматься не собирается — военные расходы уже превысили 100 млрд. упёртых енотов и продолжал плавно и решительно ползти к своему первому триллиону. Поэтому таких инвестиций, как это сделал в своё время Франклайн Рузельт, брать было уже абсолютно неоткуда. Да и Рузельт, честно признаемся, госбюджет так же предпочитал не травмировать, а использовал крайне убедительные аргументы в разговорах с местными олигархами. Как показала история, олигархи от сотрудничества с Рузельтом не просто выиграли, а нажились так, что мама не горюй. Не зря же говорят — кому война, а кому мать родна... Например, расширение транспортной инфраструктуры оказалось крайне полезной во времена второй мировой. Актёра Рональда Рейгана сравнивать с Рузельтом даже и пытаться не стоит. Тем более, что государство в военных вопросах, стараниями того же Рузельта, осталось только в одной роли — дойной коровы.

Так вот, возвращаясь к Столлману, ему эту идею явно подкинул кто-то из высших мира сего. И не только подкинул, но и поддержал, потому что Столлман немедленно уволился из MIT, где он тогда работал, и занялся своими делами. Как оказалось, идея открытых программ была весьма здравой. Научные алгоритмы в те времена, в отличие от наших, предпочитали патентами насмерть не закрывать. Таким образом, занявшихся каким-либо делом, можно было его делать не наобум, а на строго научной основе. И уж в этом случае непременно понадобятся расчёты. А вот тут-то можно и развернуться без сколько-нибудь серьёзных вложений, поскольку с наступлением эры НТР, охотников считать в уме с каждым днём становилось всё меньше и меньше — счёты, арифмометры, электрические и электронные калькуляторы, затем и компьютеры, в том числе персональные. В общем Америка смогла ещё немного подержаться на плаву... ☺

Компания Borland после появления Delphi на свой TurboPascal, откровенно скажем, плонула. Не то чтобы «бери кто хочешь», но о судебных преследованиях за его «бесплатное» использование я что-то не слышал. Другое дело, что на 32-ухразрядных ОС он, ясное дело, долго бы не протянул. Логично

предположить, что на основе TurboPascal, тем более, что его исходные коды во многом были доступны, кто-то попытается сделать 32-ухбитный клон. И приходит кума любоваться — в 1995 году появился 32-ухбитный VirtualPascal, который написал наш программист Виталий Милянов. Это была практически копия текстовой IDE с компилятором и отладчиком. Первое время он был предназначен только для Windows, потом появилась версия для OS/2 но развивалась она не очень охотно. Потом появилась некоторая, прямо скажем, довольно слабая поддержка Linux. Виталию проект через какое-то время надоел и им ещё немного занимался американский программист Аллан Иертнер. Проект заглох в 2004 году на версии 2.1. От себя хочу сказать, что это была отличная штука и если бы там можно было менять в IDE кодировку исходных текстов, продержалась бы она в сообществе дольше за счёт того, что виндовые компоненты пользовательского интерфейса можно было бы продолжать создавать в Borland'овском Resource Workshop. Но, увы... Занимать им впоследствии никто не стал, потому что такая сложная штука была написана на ассемблере. Для производительности это хорошо, а вот для поддержки и развития — никуда не годится. По всей видимости именно из-за этой причины сам Виталий отказался от дальнейшей его разработки.

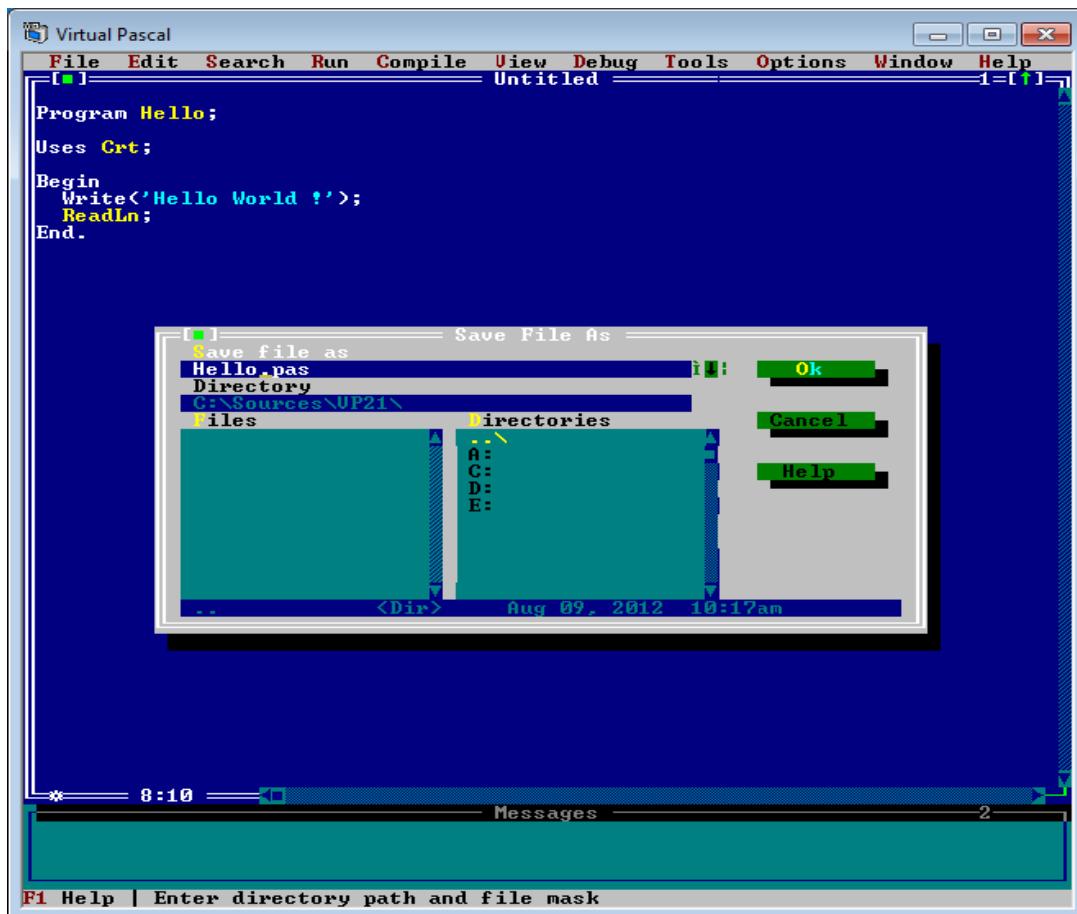


Рисунок 48. Российский Virtual Pascal Виталия Милянова

Следующим был ТМТ Pascal. Так же российская разработка — клон

TurboPascal. Когда он появился — точно никто не знает. Видимо потому что он изначально был платным, а наши люди такого не любят. Интерес представляет бесплатная версия 3.50, которая вышла в 2000-ном или 2001-ом году (сейчас точно этой даты тоже никто не помнит). Это была версия с некоторыми ограничениями и имела графическую оболочку, что популярности ей нисколько не прибавляла, т.к. Delphi уже всех приучила к визуальному программированию. Это был клон TurboPascal до версии 2 включительно только с текстовой оболочкой, а с версии 3 стал клоном for Windows, который комплектовался заголовочными файлами для DirectX, чем и был некоторое время интересен. Однако отсутствие визуального программирования популярности не способствовало. Проект был продан американской компании Framework Computers из Массачусетса (прикольно — город, где находилась компания, называется Брайтон — сразу вспоминается Вилли Токарев... ☺). Проект продержался до 2010 года и версии 6.1. Бесплатных релизов больше не было. Сам я с ним какое-то время поигрался, но поскольку DirectX для меня был неинтересен, никаких проектов на нём так и не написал.

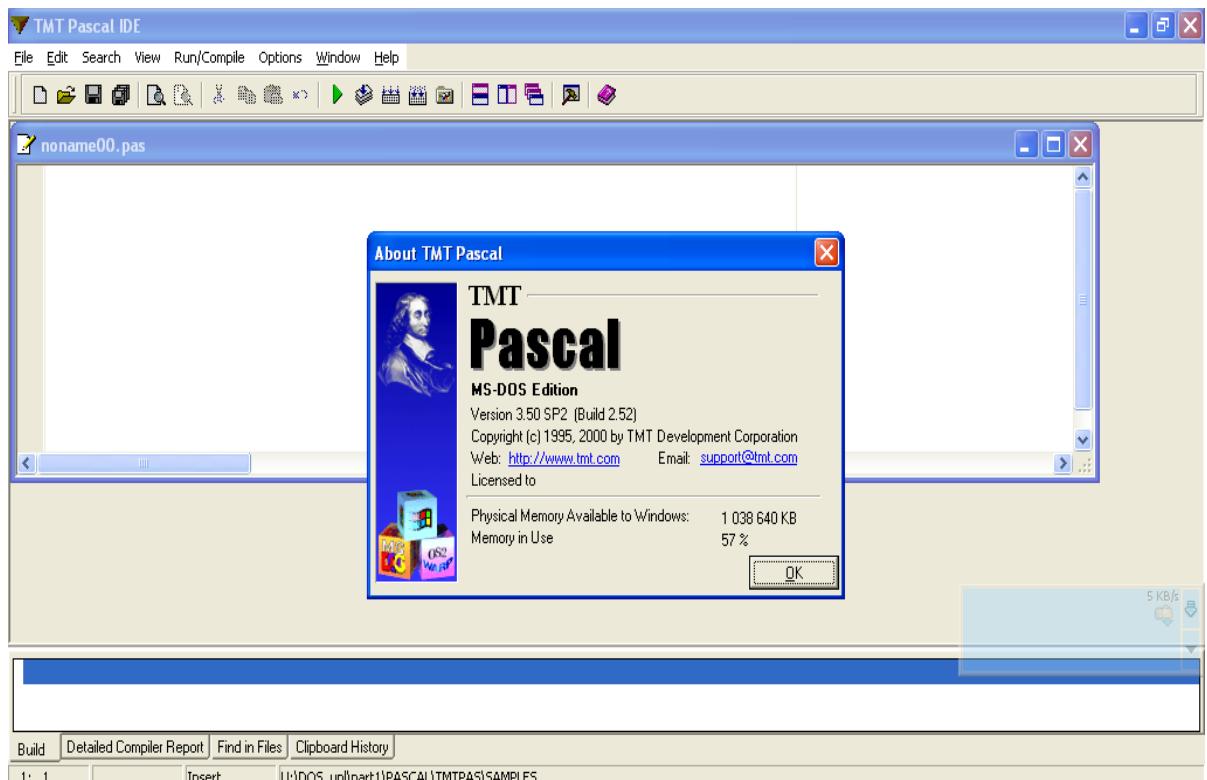


Рисунок 49. Тоже российский (обрезанный) TMT Pascal

Обойти столмановское GNU при всё желании не удастся. GNU Pascal стал известен в 1998 году, хотя разработка его началась намного раньше. Основателем GNU Pascal считается финн Jukka Virtanen, однако переносом отдельных модулей из TurboPascal стал заниматься профессор Peter Gerwinski из Эссена (Германия) ещё в 1995 году. Переносом Run-Time модулей ориентировочно в 1997 году занимался Frank Heckenbach. Перечислить всю

команду трудно, их много. Более-менее законченный и хорошо работоспособный вид GNU Pascal приобрёл в 2000-2001 году. Проектом GNU в полной мере его назвать нельзя, потому как на сайте GNU его нет, но выходил он под лицензией и патронатом именно GNU, используя его рекламные возможности. Впервые со времён, когда Паскаль першёл на «тёмную сторону силы» ☺, т.е. стал коммерческим и однопроцессорным, это был многоплатформенный Паскаль. Хотя он продолжал оставаться однопроцессорным (Intel), однако мог работать уже в нескольких ОС: Windows, DOS с 32-ухбитным расширением DJGPP или EMX, Linux, семействе BSD (Free-, Open-, Net-, MacOSD, Solaris). Довольно неплохо (но не в полной мере) он поддерживал процессоры MIPS и Sparc. Поддерживал так же 32-ух и 64-ёхбитные процессоры. Соответствовал последнему стандарту ISO-Pascal. Это была уже очень серьёзная вещь. Пока FreePascal не вышел на более-менее стабильные рельсы, я пользовался этим компилятором. Впечатления — отличные. Теоретически с ним можно работать и сейчас, однако в связи с бурным развитием FreePascal он прекратил свой жизненный цикл в 2005 году (последняя версия вышла в марте), поэтому поддерживает только 8-мибитную кодировку символов. Таким образом он может работать в Free- Open- NetBSD (но дополнительно потребуются старые библиотеки Си) или в DOS\Windows. Собственной IDE не имел, но его можно использовать совместно с каким-нибудь программистским редактором, который позволяет запускать компилятор. Ну и, разумеется, Lazarus, в котором можно выставить любую доступную кодировку исходников.

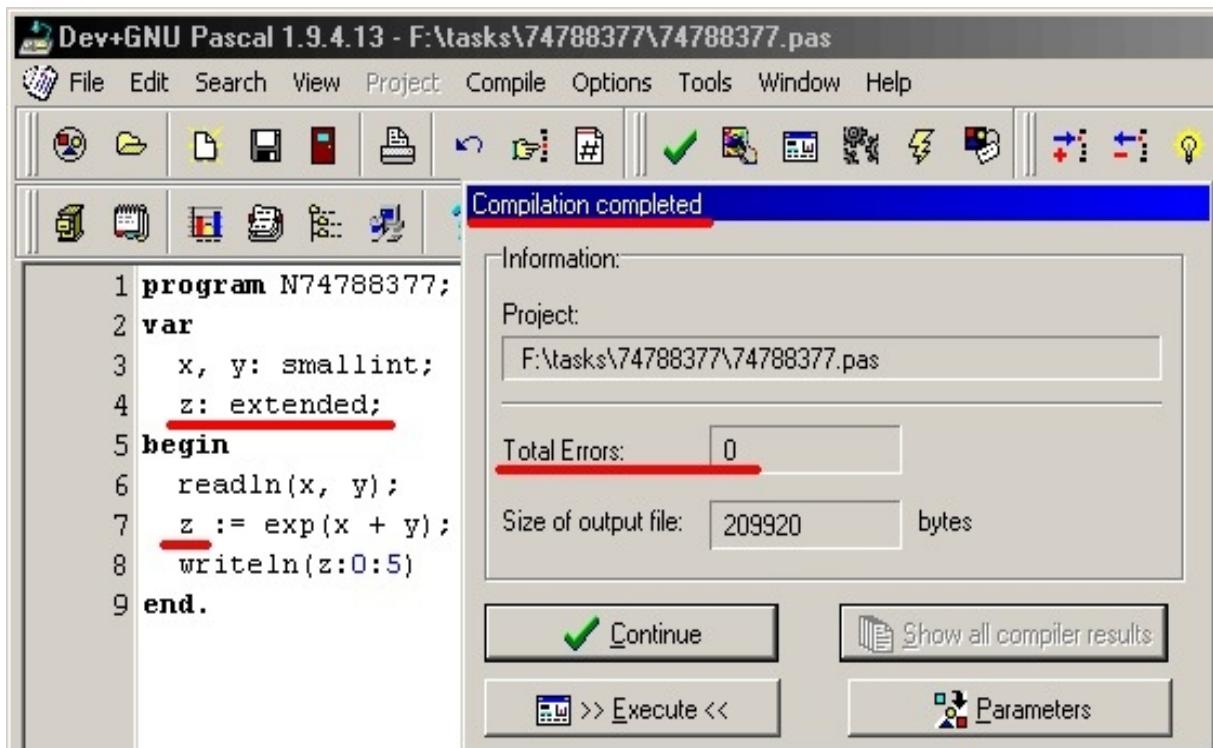


Рисунок 50. Gnu Pascal с оболочкой Dev-Pascal

Ну и, наконец, гвоздь сезона, мечта всех времён и народов — FreePascal, который остался не только в сердцах программистов. ☺ С ним я впервые познакомился примерно в 2000 году, когда искал не обременённый проприетарными лицензиями компилятор Бейсика. ☺ Широкополосного интернета у меня тогда ещё не было, но новый Паскаль я всё-таки решил скачать и попробовать. К тому времени из Паскалей я знал только TurboPascal, Delphi и VirtualPascal. Поскольку FreePascal сильно напоминал VirtualPascal, я особой разницы в их использовании не увидел. Скачивал я тогда, по-моему, версию 1.0.3 и в ней меня всё устраивало, кроме программирования графики с модулем Graph. Не то чтобы оно совсем не работало, но программа всё-таки вылетала время от времени. Впрочем, графика никогда не была моим любимым занятием в программировании, поэтому к таким «дефектам» я отнёсся вполне спокойно. Собственно и восторга FreePascal никакого не вызвал — ну, ещё один Паскаль, ничего особенного...

Особенности начались, когда я стал осваивать Linux. Тут уж со всей остротой встал вопрос — на чём программировать. Понятно, что ни TurboPascal, ни Delphi в Linux не засунешь (Wine тогда отсутствовал как явление). GNU Pascal в состав какого бы то ни было тогдашнего дистрибутива не входил, да и у него в то время тоже были свои косяки, хотя и тоже непринципиальные. Выбор FreePascal произошёл потому, что в Windows GNU Pascal работал несколько похоже, чем FreePascal. К сожалению, сейчас не помню в чём там у GNU Pascal была проблема, но выбор был однозначный — FreePascal, потому что он работает и в Windows, и в Linux. Бонус — не надо держать в голове наборы опций от разных программ.

В версии 1.0.10 проблемы с модулем Graph для Windows наконец-то исправили и с тех пор FreePascal я считаю нормальным Паскалем. Тогда же я полностью забросил Delphi. Поскольку я нашёл FreePascal, то довольно быстро отыскалась и графическая IDE Lazarus. Сейчас точно не помню, какая это была версия, по-моему 0.2. Несмотря на то, что она ещё не вышла в релизы, т.е. не достигла версии 1.0, программировать окошечные приложения в ней мне понравилось. Сначала речь не шла о программировании окошек в Linux, поэтому кодировка 1251 в той старой версии Lazarus меня ни капли не смущала. Да, были иногда кое-какие проблемы, но обычно они решались без особого напряжения мозга, что меня в Lazarus и подкупало. После перехода Lazarus на UTF-8 (версия 0.9.25), стали посещать мысли и об окошечном программировании в Linux, поскольку Lazarus позволял не изучать, ломая мозг, все эти гномы и кутэшки.

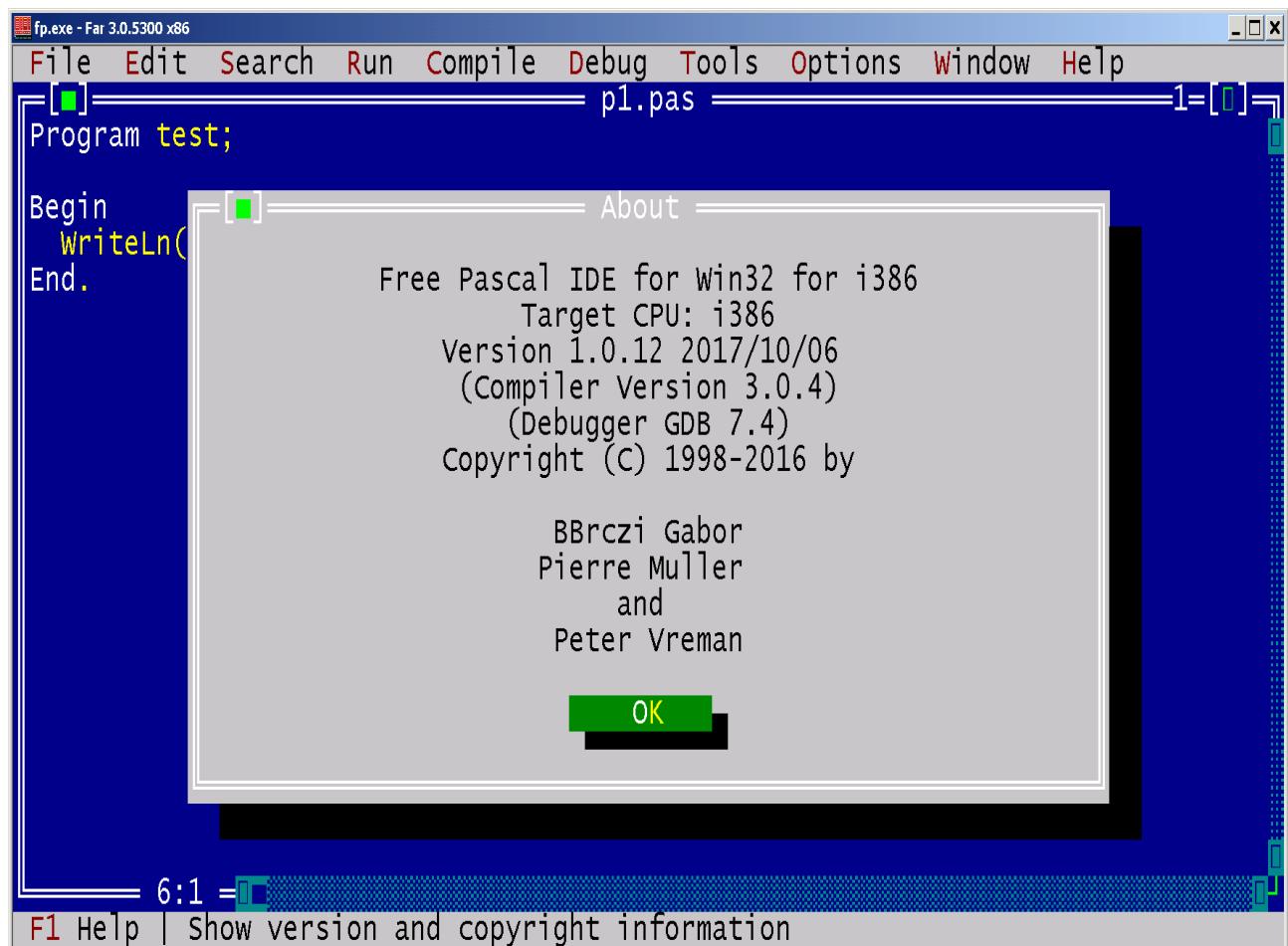


Рисунок 51. FreePascal и его текстовая IDE

Развитие FreePascal и Lazarus идёт эволюционным путём. Пожалуй революция была только однажды — в Lazarus сменилась кодировка текстов по умолчанию на UTF-8. Но это было не особо революционно, поскольку позволило слить два параллельных проекта — виндовый и линуксовый Lazarus в один, что сильно облегчило его совершенствование.

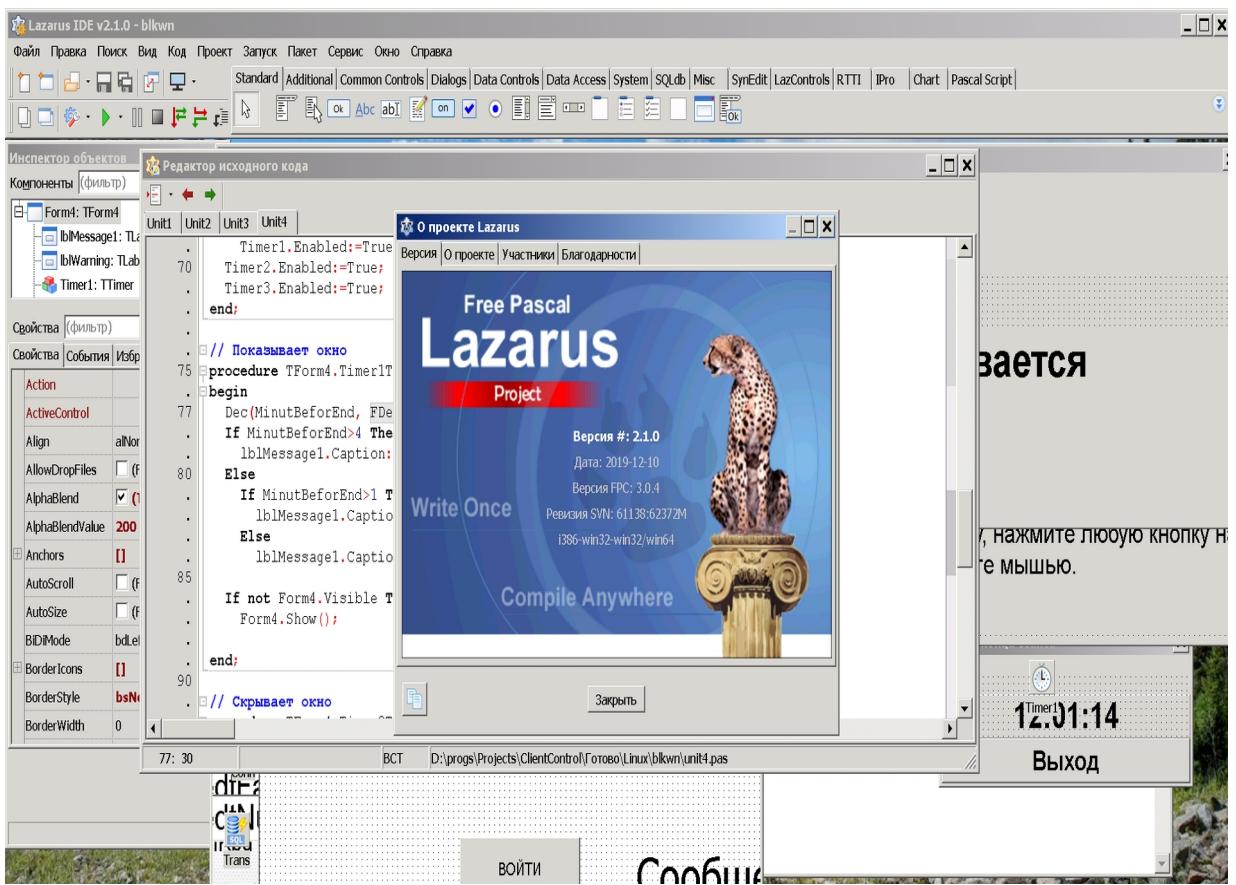


Рисунок 52. Графическая IDE для FreePascal — lazarus

FreePascal можно оснастить тремя графическими IDE, позволяющими программировать визуально:

- **Lazarus**, наиболее часто используемая IDE. Представляет собой надстройку верхнего уровня над элементами графического интерфейса, которые уже установлены в системе — Win32, WinCE, GTK, Qt. Принципы работы — такие же, как и в Delphi 7. Релизы этой IDE выходят нечасто, однако разработчики каждый день на своём ftp-сервере [26] выкладывают снапшоты всей системы в виде исходников. Может работать на всех ОС, куда удастся поставить FreePascal и какую-нибудь графическую библиотеку элементов, типа GTK или Qt. Очень большое количество визуальных элементов, но это может быть недостатком на маломощной системе, т.к. потребляет много ресурсов. Работает подсказка кода, как в Delphi, что очень удобно для тех, кто не хочет забивать голову техническими подробностями компонентов или функций;
- **MSEIDE** [27], IDE в которой элементы строятся не на каких-то существующих, а отрисовываются самостоятельно, используя xlib в Linux или GDI в Windows. Очень удобная штука, если надо чтобы ваше приложение выглядело 100% одинаково и в Windows, и в Linux. Система более легковесная, т.к. не использует никакие дополнительные

библиотеки графических элементов. Элементов много. На сегодняшний день, в связи со смертью основного разработчика, дальнейшая судьба этой IDE пока не ясна;

- **UIDESIGNER** [28], так же как и предыдущая, отрисовывает графические элементы самостоятельно. В отличие от MSEIDE, работает как генератор Паскаль-кода, т.е. не использует графические ресурсы для элементов, а просто создаёт текстовый исходник. С помощью неё можно создавать наиболее легковесные оконечные приложения, т.к. получившийся исходник можно дополнительно оптимизировать. Обновляется очень редко.

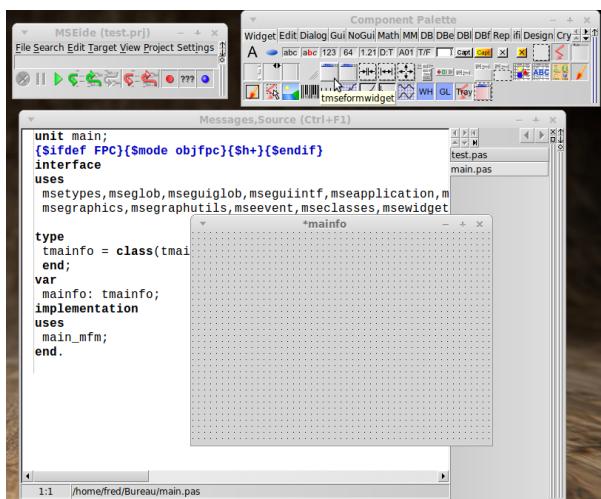


Рисунок 53. Графическая IDE MSEIDE

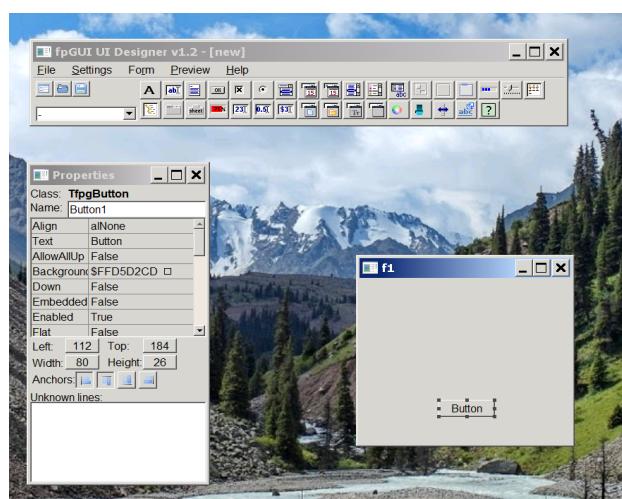


Рисунок 54. Графическая IDE UIDESIGNER

Если говорить в общем, то FreePascal представляет собой невизуальную часть Delphi, а визуальная это отдельный проект — Lazarus. В FreePascal отсутствует часть компонентов, которые представляют собой специализированные Windows-компоненты, например, для работы с IE. Основная цель проекта — сделать незаметным миграцию программ на ту или иную платформу. Если использовать общие компоненты, которые не OS-специфичны, то такая задача не представляет трудностей. В иных случаях, когда в код нужно помещать некие специфичные для той или иной ОС функции, то можно пользоваться элементами условной компиляции, которые заносятся в код программы, например:

```
{ Получение имени текущего пользователя }
function GetLocalUserName: string;
begin
  {$IFDEF WINDOWS}
    result:=GetEnvironmentVariable('USERNAME');
  {$ELSE}
    result:=GetEnvironmentVariable('USER');
  {$ENDIF}
end;
```

Ответвление — PascalABC.NET

Это специализированный учебный проект [29], который изначально создавался, как замена устаревшему TurboPascal для учебных курсов по программированию на факультете математики, механики и компьютерных наук Южного Федерального Университета. Первые версии (а появился он в 2003 году) назывались просто PascalABC. Изначально это была оболочка со встроенным интерпретатором языка Паскаль. В дальнейшем, примерно к 2006 году, его решено было перенести на платформу .NET и дополнить компилятором. На первый взгляд в программировании после этого ничего не поменялось, однако от традиционного виртовского Паскаля там отошли уже довольно далеко. Например, прямо в теле программы можно объявлять переменные. Но есть и явные преимущества по сравнению с традиционными и современными Паскалями, например, возможность использовать в коде многопоточное программирование с помощью OpenMP, которое значительно проще и легче, чем с помощью функций ОС. Поэтому можно буквально за 10 минут переделывать сишиные или форTRANовские многопоточные программы на наш любимый Паскаль. Студенты от такого будут в полном восторге. ☺

Работает исключительно на тех ОС, где установлен Microsoft .NET Framework v4.7, т.е. это Windows 7 и выше или 2008 и выше. Встроенных модулей-библиотек немного, потому что можно использовать любые пространства имён .NET в строке Uses. Из собственных может представлять интерес NumLibABC — численные методы или WPFOObjects — векторная графика.

Код программ на традиционный Паскаль может быть похож только отчасти, поэтому здесь надо быть очень внимательным:

```
//Сколько нечетных среди n введенных

begin
var n:=ReadInteger('Введите n: ');

var c:=0;
for var i:=1 to n do
begin
  var x := ReadInteger('Введите целое число: ');
  if x mod 2 <> 0 then
    c += 1;
end;

writeln('Количество нечетных равно {c}');
end.
```

На сегодняшний день это аналог Delphi for .NET, однако, на мой взгляд, более удачный, поскольку позволяет писать код программ в традиционном

паскалевском стиле и, одновременно, использовать большое количество уже написанных библиотек .NET. Лицензия — свободная, LGPL.

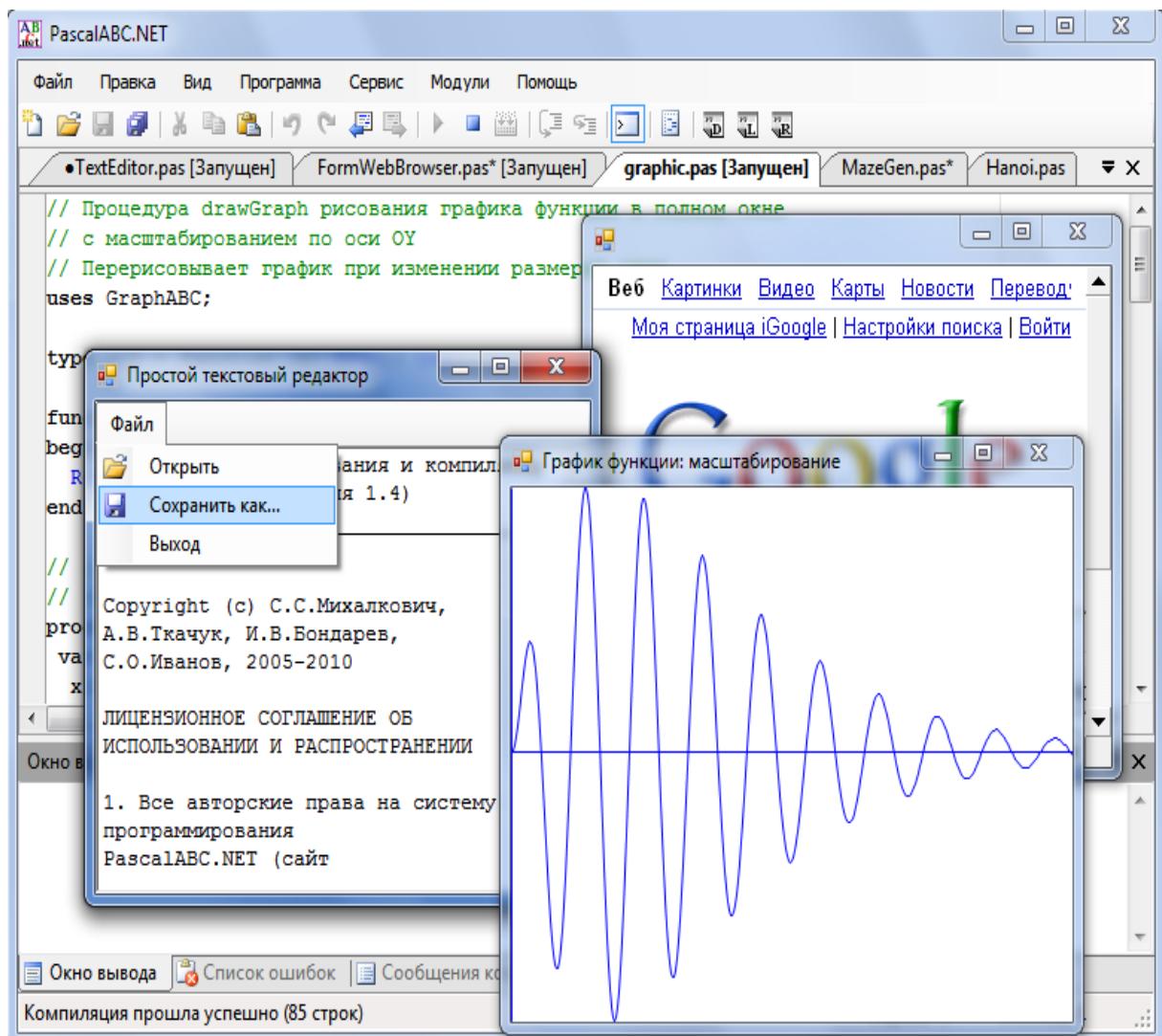


Рисунок 55. PascalABC.NET и его IDE

PascalABC.NET и Linux. Тут есть два варианта:

1. Работать в стандартной графической оболочке через WINE. Каких либо отличий от работы в Windows заметить будет крайне трудно. При установке скачается последняя версия .NET, если до этого она не была установлена;
2. Консольный компилятор rabcnetc.exe, который работает через MONO. Предварительно нужно установить метапакет «mono-complete», чтобы потом не чесать тыковку, чтобы могла означать та или иная NET-ошибка выдаваемая при компиляции вашей программы. После компиляции вы получите EXE-программу и если ваша оболочка уже настроена запускать такие файлы через WINE или MONO, то больше никаких телодвижений делать не придётся.

Что интересно, по умолчанию этот компилятор будет разговаривать с вами на украинской мове... ☺

```
vadim@mycomp1:~/progs/PascalABC.NET$ mono pabcnetc.exe p1.pas
Завантажую ядро...
Підключений парсер PascalABC.NET Language Parser v1.2
Підключений парсер Documentation Comments Tag Parser v0.9
PascalABCCompiler.Core v3.6.3.2461
Copyright (c) 2005-2020 by Ivan Bondarev, Stanislav Mikhalkovich
OK 127,996ms
Підключені парсери: PascalABC.NET (*.pas);
Підключені перетворення: Optimizer;
Компілюю збірку p1.pas...
OK 1561,356ms (8рядків)
vadim@mycomp1:~/progs/PascalABC.NET$ ./p1.exe

Здравствуй твоя-моя...
Моя сделана через консольный pabcnetc, однако...

vadim@mycomp1:~/progs/PascalABC.NET$ mono p1.exe

Здравствуй твоя-моя...    I
Моя сделана через консольный pabcnetc, однако...
```

Рисунок 56. Программа собрана в Linux с помощью консольного компилятора

Стоит ли изучать PascalABC.NET? Если это предписано вашими учебными программами — безусловно да. А вот на воле о каких-либо проектах (например, WindowsABC.NET ☺) я что-то и не припомню. PascalABC.NET даст вам отличную базовую подготовку по программированию, и, в отличие от курсов по C\C++, более понятную. Вдобавок, вам потом никто не будет мешать с лёгкостью использовать уже готовые многочисленные библиотеки .NET, даже если они написаны на других языках — платформа всё равно одна и та же.

Дети Паскаля — MODULA

В 1976 году Вирт высыпал в своём институте годичную командировку в XEROX PARK, увидел там много интересного, а в числе прочего — графическую рабочую станцию «Xerox Alto». Попытавшись купить её для своего института, он наткнулся на вежливый, но твёрдый отказ. Как ему пояснили, против был не XEROX, а чуть ли не сам американский КОКОМ (комитет по экспортному контролю). Объяснение его, мягко говоря, озадачило — дескать, поскольку Вирт большой друг Советского Союза, то КОКОМ опасается, что он немедленно сольёт Советам эту рабочую станцию и те, коварные, узнают из неё все самые секретные американские секреты. ☺ Мне это напомнило фильм с Луи де Финесом «Жандарм и жандарметки», где в жандармерию Сан-Тропе нечаянно попал суперсекретный компьютер с суперсекретными данными. ☺ Естественно Вирт на такой демарш обиделся и решил: «Я вам покажу жандарметок! Не продаёте — сами сделаем!». ☺



Рисунок 57. Секреты в XEROX'e?????!!!! ☺

Небольшое поверхностное расследование выявило, что никакой КОКОМ там ни сном, ни духом. На самом деле какого-нибудь более-менее существенного производства систем «Xerox Alto» компания так и не смогла организовать. Вручную собиралось некоторое количество этих компов исключительно для внутреннего использования. Естественно, от зоркого глаза Вирта этот пикантный нюанс не укрылся, поэтому возникла идея делать то же самое — т.е. собирать на месте, в Цюрихе, собственные системы из местных

запчастей. Тем более, что практику по электронике у студентов никто не отменял... ☺

Сказано — сделано! В 1979 году был собран экспериментальный вариант компьютера «Lilith», который проектировался по виртовской концепции: язык и ОС — едины! В качестве системного языка был реализован вариант языка Паскаль — Модула-2. А где же Модула-1 спросите вы? Модула-1, как и первоначальный вариант Паскаля, служил у Вирта полигоном по отработке языковых концепций, в частности параллельного программирования. Поэтому официально он Виртом не заявлялся. Первые, экспериментальные, версии языка были реализованы на мини-ЭВМ PDP-11. «Мини» она называлась потому, что очень скромно занимала всего «один шкафчик», т.е. одну 19-тидюймовую стойку и имела довольно простую систему процессорных команд. Благодаря своей простоте и минимализму в размерах, это была очень популярная машина, продавалась широко и последние её разработки-модернизации датируются 1990-ом годом, правда уже на микропроцессорах, а не на простых логических микросхемах. В СССР она так же реинкарнировалась в большом виде — серия «СМ», а в маленьком — серия «Электроника» и «ДВК».



Рисунок 58. Сравните XEROX Alto (слева) и DISER Lilith (справа). Похожи? ☺

Многих может удивить странная ориентация дисплея. На самом деле тут нет ничего странного. Дисплей и у Alto и у Lilith ориентирован на стандартный документ Letter (или по нашему А4), книжного формата. Компьютеры собирались силами студентов и выпущено их было несколько сот штук. Большинство экземпляров осталось в институте для учебной и научной работы.

Из интересного. Электроникой и микропрограммами для процессора занимался Ричард Оран, который учился в том же институте. Системный блок Lilith был четырёхпроцессорным за счёт чего обеспечивалась параллельность вычислений. Что интересно, все 4 16-тибитных процессора работали на одну шину, но шину специфическую — 64-ёх разрядную, т.е. у каждого процессора был свой участок шины данных. В Alto шина была 16-тиразрядной и поэтому его единственный процессор, сильно парился с обработкой графики. У «Lilith», за счёт широкой шины, и многопроцессорности нагрузка при обработки графики уменьшилась до 3% (по сравнению с 50% у «Xerox Alto»). Процессор был не однокристальный, а сборный, на мелкой логике.

Советские аналоги PDP-11



Рисунок 59 DVK-3. Такой стоял в нашем техникуме и мы на нём изучали Бейсик



Рисунок 60. Электроника-60М. Почти суперкомпьютер, на нём можно было играть игрушки... ☺

В отличие от Паскаля, Модула стал языком регистров зависимым. Сделано это специально для большей понятности текстов программ и повышения скорости компиляции. В отличие от Си, в Модула все ключевые слова положено писать в верхнем регистре. Названия объектов, сделанные программистом, далее писались именно в том виде, в каком были предварительно описаны. Блоки кода модернизированы таким образом, чтобы иметь явно видимое начало и конец действия. К примеру, циклы (операторы IF) и прочие подобные штуковины в обязательном порядке должны заканчиваться ключевым словом END. Я полагаю, что эта идея была взята из Бейсика. Впоследствии такое же правило ввели и в стандарт Фортран-90.

Своё название — Модула, язык получил из-за того, что теперь программа

— это не один единственный файл, а набор отдельных модулей. Это благоприятно повлияло на время сборки больших программ — компилировались только ранее неоткомпилированные (или изменённые) модули. Правда модули сделали похожими на Си — отдельно модуль заголовков (экспорт) и отдельно модуль реализации. Это оказалось не очень удобным, поэтому в следующем за Модулой Обероне от подобного принципа отказались — остался один модуль реализации, где специальным образом помечалось, что идёт на экспорт.

Тип «строка» стал похож на сиший, т. е. это стал массив символов (ARRAY OF CHAR) который заканчивается нулём («0X», по сициальному — «\0»). Таким образом стало можно иметь «бесконечные» строки, не ограниченные паскалевским размером в 256 символов.

Некоторые упрощения, которые Вирт ввёл в Модулу, нам могут показаться странными. Например, вы нигде не найдёте упоминания термина «функция». А их там и правда нет, остались одни процедуры. Вы спросите, а как же вычислять синус-косинус? За ответом далеко ходить не придётся. Можно, к примеру, заглянуть в Фортран. В нём хоть такая программная единица как «функция» имеется, однако в подавляющем большинстве случаев программисты предпочитают пользоваться процедурами или, по фортраньи, подпрограммами. Почему? А вот вам ответ — функция может вернуть в главную программу только одно значение, а процедура — сколько угодно. Или точнее будет сказать, сколько можно запихнуть параметров в заголовок процедуры. Правда в Фортране при виде иных процедур/подпрограмм могут волосы дыбом встать — чуть ли не два десятка параметров... ☺

Впрочем, это я вас просто пугаю, на самом деле функция, как программная единица, всё же осталась. Теперь процедура может быть функцией и выглядеть, в таком случае, следующим образом:

```
PROCEDURE InCube(Number1 : INTEGER) : INTEGER;
BEGIN
  RETURN (Number1 * Number1 * Number1);
END InCube;
```

Т.е. в хвост заголовка добавляется возвращаемый тип, а в теле процедуры вводится ключевое слово RETURN, у которого в скобках указано возвращаемое значение или выражение. RETURN'ов может быть несколько, например, в случае разветвляющегося результата, но всегда RETURN (возвращаемое_значение) — это последнее, что делает процедура-функция.

В отличие от языка Паскаль, в Модуле (а ниже и в Обероне), для процедур и функций использование скобок является строго обязательным, даже если вы не передаёте процедуре никаких параметров. С одной стороны это сделано для программиста, чтобы в тексте программы процедуры были явно заметны, а с другой стороны — для компилятора, чтобы тот не терял зря времени копаясь в своих многочисленных и крайне разветвлённых

справочниках, пытаясь выяснить, а что же на этот раз ему программист подсунул — переменную или процедуру...

Модула — один из языков, где появилось упоминание про параллельное программирование. Параллельное с помощью отдельных процессов, но поскольку процессы работают в едином адресном пространстве, сейчас их нужно называть потоками или нитями. И, скажем честно, не то чтобы очень то параллельное. ☺ Параллельность здесь — это возможность запустить отдельный процесс-поток и возможность передачи ему управления. В принципе, это и всё, никаких языковых конструкций для обмена данными нет. Наверное Вирт для единого адресного пространства посчитал это излишним — данные можно передавать через экспортируемые переменные.

На сегодняшний день упоминаний языка Модула в интернете очень много, однако 99.99% это перепечатка Википедии в той или иной степени полноты. Упоминаются языки Модула-2 и Модула-3. В реальности же остаётся активен только Модула-2 ([30] и [31]). Для него даже есть международный стандарт [32]. Почитать стандарт на русском языке можно здесь [33].

Модула-3 скорее мёртв, чем жив. Дело в том, что это модифицированная коммерческая, а потом и открытая версия языка от фирмы DEC, которую они разрабатывали совместно с итальянской компьютерной фирмой Olivetti. Один из разработчиков DEC в 1986 году некоторое время переписывался с Виртом по поводу модернизации Модула-2, однако Вирт, который в то время уже был плотно занят Обероном, хоть и проявил интерес к модернизации, но сам заниматься Модулой не собирался. Таким образом Модула-3 можно считать не новой версией, а скорее внутрифирменой модернизацией Модула-2. После того, как DEC прекратила своё существование, серьёзного разработчика для Модула-3 так и не нашлось. Проект переходил из рук в руки, побывал коммерческим, потом стал открытым. После выхода в OpenSource у Модула-3 появился свой собственный сайт [40], однако последняя активность там наблюдалась аж в далёком 2010 году и с тех пор у них стоит гробовая тишина.

Кардинальных отличий у Модула-3 от Модула-2 целых два:

1. Явное введение в описание ООП в виде ключевого слова **OBJECT**. Правда работать этот **OBJECT** продолжает так же, как и **RECORD**. Отличие от **RECORD** в том, что **OBJECT** всегда является указателем, так же как и **CLASS** в Delphi;

2. Включение в описание языка ловли исключительных ситуаций **TRY ... EXCEPT ... END** и **TRY ... FINALLY ... END**.

В остальном всё примерно одинаково. Так что если подумать, никто бы особо новым языком и не заинтересовался. Однако, кроме языковых изменений, были изюминки в булке — ряд стандартных библиотек, которые делали программирование удобным, вот например:

- **Набор библиотек GUI** — VBTKit, FormsVBT и FormsEdit. Конечно к языку эти библиотеки имели отношение постольку-поскольку, однако в своё время программисты их восприняли на «Ура!», потому что это были библиотеки верхнего уровня, так что построить с их помощью GUI было легко, не то что на WinAPI. Правда есть тут один минус для пользователей Windows — элементы интерфейса отчего-то очень сильно напоминали своих братьев в юниксовой графической оболочке Motif... ☺ IDE Модула-3 была построена на этих библиотеках. Ходят слухи, что у разработчиков были планы сделать верхний уровень интерфейса как компоненты в Lazarus, т.е. привязываемые к какой-то определённой графической библиотеке. Но, не сложилось...

- **Сетевые библиотеки**, которые позволяли запускать программы в сети, как в едином адресном пространстве. Представьте себе, что вам стало не хватать для вычислений ресурсов своего компьютера. Ничего страшного, компьютер соседа тоже подойдёт, причём не требуется для этого вводить какие-то дополнительные алгоритмы для взаимодействия данных. Правда было тут ограничение — у создаваемого сетевого объекта не должно быть полей данных. Сначала это кажется непривычным по нынешним временам, но если маленько подумать — ничего сложного, можно вспомнить IUnknown и его наследников. ☺

Если вы захотите поковыряться с дистрибутивом Модула-3, то вероятность, того что он заработает не очень велика, т.к. строится на старых системных библиотеках. Но если очень уж интересно, то можно попробовать пошаманить с softlinksами, возможно и удастся его запустить.

Модула-2 продолжает развиваться. Наверное наиболее популярным экземпляром на сегодняшний день можно назвать Модула-2 от проекта GNU [34] под названием «gm2». Точнее говоря это фронт-энд для GCC. Популярности способствовало то, что и Debian, и Ubuntu в своих последних версиях включили этот пакет в свои репозитории. Правда в официальный комплект GCC gm2 пока что не попал. Есть сведения, что в LLVM идут дискуссии, нужна им Модула-2 или не нужна. На ГитХабе лежат исходники для компиляции совместно с LLVM, но судьба этого проекта неясна. У LLVM похоже на тщательное курирование всех интересных разработок не хватает ресурсов, в отличие от GNU. Примером тому может служить Фортран, LLVM-ная версия которого появилась очень давно, но в стандартный комплект так и не попала. Так что у gm2 больше шансов обрести себя в GCC...

В официальных репозиториях Debian бинарный пакет есть начиная с версии ОС 9.2, в Ubuntu — для версии ОС 19.10. У Debian пакет пока что сидит в нестабильном репозитории, но он рабочий, просто не до конца проверенный.

Ещё один крайне интересный экземпляр — от российской фирмы Эксельсиор [37]. Это разработка велась в новосибирском вычислительном

центре (сейчас — Институт Систем Информатики им. Ершова) по заказу НПО Прикладной Механики, Красноярск, для программирования бортовых ЭЦВМ спутников. Проект начался ещё в конце 80-ых, однако был приостановлен и возобновлён в начале 2000-ых, с новыми спутниками ГЛОНАС.

Эксельсиорский продукт интересен тем, что он универсальный — может работать и с языком Модула-2, и с языком Оберон-2. Непосредственно в двоичный код он не компилирует, делает сначала промежуточный вариант на Си, а потом уже идёт компиляция сишного кода. Однако не думайте, что его можно использовать как утилиту «Модула2Си», сишный код на выходе получается совершенно непонятный, хотя и читаемый. ☺

Кстати говоря, новосибирцы в своё время спроектировали и изготовили процессор, и рабочую станцию «Кронос», аналог Lilith, только 32-ухразрядные.



Рисунок 61. Процессор Kronos P2.5

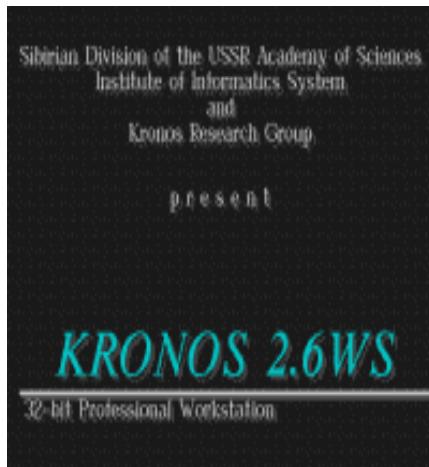


Рисунок 62. Операционная система



Рисунок 63. Рабочая станция КРОНОС-2.6WS

Рабочая станция КРОНОС использовалась на НПО ПМ в качестве машины для создания бортового программного обеспечения спутников. На ней стояли компиляторы Си, Модула-2, Паскаль и Оккам.

Перспективы языка. Разговаривать будем только о Модула-2. Наверное, в первую очередь, возникнет вопрос — а как там с ООП? Про ООП в Модуле Вирт написал ещё в 1989 году в своей статье в журнале «Microprocessors and Microsystems» за апрель 1990 год. Перевод на русский можно почитать здесь [41]. Я, откровенно говоря, не очень понимаю всю эту ажиотацию по поводу ООП, но раз надо значит надо. ☺ В Модуле ООП может работать частично. Есть составной тип данных RECORD, в него можно включать процедурные типы. Можно использовать ссылочные типы, т.е. создавать RECORD'ы динамически, по мере возникновения потребности. И, в принципе, по ООП всё. ☺ Во всём остальном — это хороший и, по сравнению с Ада, несложный язык.

Для создания окошечных приложений нужно смотреть конкретный транслятор, есть ли у него интерфейсы к функциям GUI-библиотек, типа GTK или WinAPI [42] и т.п. У Эксельсиоровского варианта такие интерфейсы есть.

Программирование на Модула-2. Приготовьтесь друзья мои к тому, что когда вы садитесь за новую систему программирования, она вам будет много и противно нудеть о том, что вы всё делаете не то и не так... ☺

В первую очередь хочу предупредить, что в Модула-2 (и далее в Обероне), по сравнению с Паскалем поменялись символы двух операций:

- Отрицание (NOT). Обозначается символом «~» (тильда);
- При сравнении "не равно". Обозначается символом «#» (решётка).

Естественно, что названия модулей будут другими, да и названия часто употребительных функций тоже. ☺ Я решил на себе испытать, насколько фатальна для моих нервов будет резкое пересаживание на Модула-2. Код примерчика я взял отсюда [43], поскольку меня интересовало не столько программирование само по себе, сколько совместимость старинного кода, найденного в интернете (которого намного больше, чем современного ☺), с новыми компиляторами. В качестве компилятора я поставил убунтовский пакет «gm2». И понеслось... ☺

```
vadim@mycomp1:~/projects/modula$ gm2
Команда «gm2» не найдена, но может быть установлена с помощью:
sudo apt install gm2

vadim@mycomp1:~/projects/modula$ vadim@mycomp1:~/projects/modula$ gm2
gm2: fatal error: no input files
compilation terminated.
vadim@mycomp1:~/projects/modula$ gm2 areas.mod
areas.mod:1:2: error module name at beginning 'areas' does not match the name at end 'Areas'
areas.mod:105:1: error the file containing the definition module 'Terminal2' cannot be found
areas.mod:105:1: error module name at end 'Areas' does not match the name at beginning 'areas'
vadim@mycomp1:~/projects/modula$ █
```

Рисунок 64. Первая компиляция кода примера. Ошибки ожидаемы...

Оказывается «gm2», который я только что установил, не обнаружен. Ну, с этим я не раз сталкивался — современные ОС позволяют ставить компиляторы\интерпретаторы нескольких версий, поэтому у них у всех к имени прибавляется ещё и номер версии. Необходимо сделать симлинк для имени по умолчанию на правильный файл компилятора. После этого компилятор стал работать, но теперь пришла пора ловить ошибки в коде.

Ошибки синтаксиса выводятся красиво — расцвеченные разным цветом, мне понравилось. Первая и последняя ошибки исправляются в 5 секунд — нужно чтобы название файла строго соответствовало названию модуля. Это ожидаемо, Ада и современный Фортран реагируют так же. FreePascal с юнитами тоже. А вот вторая — отсутствие модуля, это уже серьёзно, это явная несовместимость. Поэтому настало время посмотреть, какие модули есть у Модулы «gm2» и где они лежат вообще. Чисто логическим путём стало понятно, что раз «gm2» строится на «GCC», то его модули должны быть там же, где и gcc-овские, конкретно в моём случае — «/usr/lib/gcc» и где-то там далее. Действительно, если посмотреть на рисунок ниже, так и оказалось.

```
vadim@mycomp1:~/projects/modula$ gm2 -flibs=pim Areas.mod
Areas.mod:89:9: error not expecting a string constant as an operand for either comparison or binary operation
Areas.mod:90:9: error not expecting a string constant as an operand for either comparison or binary operation
Areas.mod:91:9: error not expecting a string constant as an operand for either comparison or binary operation
Areas.mod:92:9: error not expecting a string constant as an operand for either comparison or binary operation
Areas.mod:93:9: error not expecting a string constant as an operand for either comparison or binary operation
Areas.mod:100:4: error not expecting a string constant as an operand for either comparison or binary operation
vadim@mycomp1:~/projects/modula$  
  
vadim@mycomp1:/usr/lib/gcc/x86_64-linux-gnu/9/m2/pim$  
  
vadim@mycomp1:~/projects/modula$ gm2 -flibs=pim Areas.mod
Areas.mod:97:10: error attempting to pass (1) parameters to procedure (WriteInt) which was declared with (2) para
meters
Areas.mod:97:10: error attempting to pass (1) parameters to procedure (WriteInt) which was declared with (2) para
meters
vadim@mycomp1:~/projects/modula$  
gm2 -flibs=pim Areas.mod
/usr/bin/ld: невозможно найти -lpth
collect2: error: ld returned 1 exit status
vadim@mycomp1:~/projects/modula$
```

Рисунок 65. Ошибки из-за несовместимости со старым кодом

Правда появились непонятки: в месте хранения модулей — «/usr/lib/gcc/x86_64-linux-gnu/9/gm2/», модули разбиты по каталогам и в каждом каталоге лежат файлы с совпадающими именами. Как оказалось после небольшого расследования, «gm2» предлагает варианты подключения набора модулей с помощью ключа «-fXXX», где «XXX» — это название каталога. Каталог «pim» — это набор модулей по виртовскому описанию из книги «Programming in Modula-2» (PIM). В этом случае компиляция программы будет выглядеть так:

```
gm2 -fpim MyProg.mod
```

«pim» может быть разных модификаций, по числу виртовских изменений:

«pim», «pim2», «pim3», «pim4». За подробностями обращайтесь к Вирту. Каталог «iso» — это ISO-шный стандарт Модулы-2. Компиляция с ISO-шными модулями будет такая:

```
gm2 -fiso MyProg.mod
```

ISO-шная версия считается наиболее полной, потому что там есть встроенный тип COMPLEX, плюс разные системные вкусняшки, без которых порою не обойтись.

Каталог «ulm» — это библиотечные модули взятые из транслятора Модула-2, который когда-то разрабатывался в университете города Ульм, в котором родился Альберт Эйнштейн. Кстати, этот универ как-то раз претендовал, чтобы ему присвоили имя Эйнштейна, но что-то у них там с этим делом не задалось.

В каталогах модули в виде исходников, поэтому можно отыскать необходимый модуль по использованной в программе функции контекстным поиском. Надо сказать, что в отличие от Паскаля, у Модулы собственных встроенных функций практически нет, ну может быть с десяток... Поэтому ни ввести данные, ни увидеть результатов расчётов без дополнительных модулей не удастся. Принцип подключения модулей так же отличается от Паскаля — в Паскале модуль подключается целиком, а в Модула-2 обычно из модуля подключаются конкретные функции или какие-то существующие там объекты. Это похоже на то, что есть в Питоне. Однако можно подключить модуль и целиком. При этом нужную функцию в своей программе можно будет использовать только в комплекте с именем модуля:

```
IMPORT StrIO;  
...  
BEGIN  
...  
StrIO.WriteString("Привет, чуваки!");  
StrIO.WriteLine;  
...  
END.
```

Итак, контекстный поиск выявил, какие конкретно модули нужно подключить вместо отсутствующего модуля «Terminal2». Для «gm2» их нужно несколько:

1. Для ввода\вывода строк — StrIO;
2. Для ввода\вывода целых чисел — NumberIO;
3. Для ввода\вывода нецелых чисел — FpuIO.

Ещё оказалось, что в «gm2» конструкция CASE не хочет делать выбор на


```

WriteString(" попугаев");
WriteLn;
END AreaOfSquare;

(* ***** Area Of Rectangle *)
(* Площадь прямоугольника *)
PROCEDURE AreaOfRectangle;
VAR
  Width, Height, Area : REAL;

BEGIN
  WriteString("Прямоугольник:");
  WriteString(" Введите ширину : ");
  ReadReal(Width);
  WriteLn;
  WriteString(" Введите высоту : ");
  ReadReal(Height);
  Area := Width * Height;
  WriteString("Площадь равна : ");
  WriteReal(Area,15,7);
  WriteString(" попугаев");
  WriteLn;
END AreaOfRectangle;

(* ***** Area Of Triangle *)
(* Площадь треугольника *)
PROCEDURE AreaOfTriangle;
VAR
  Base, Height, Area : REAL;

BEGIN
  WriteString("Треугольник:");
  WriteString(" Введите основание : ");
  ReadReal(Base);
  WriteLn;
  WriteString(" Введите высоту : ");
  ReadReal(Height);
  Area := 0.5 * Base * Height;
  WriteString("Площадь равна: ");
  WriteReal(Area,15,7);
  WriteString(" попугаев");
  WriteLn;
END AreaOfTriangle;

(* ***** Area Of Circle *)
(* Площадь круга *)
PROCEDURE AreaOfCircle;
VAR
  Radius, Area : REAL;

BEGIN
  WriteString("Круг:");
  WriteString(" Введите радиус : ");
  ReadReal(Radius);
  WriteLn;
  Area := 3.141592 * Radius * Radius;
  WriteString("Площадь равна : ");
  WriteReal(Area,15,7);
  WriteString(" попугаев");
  WriteLn;

```

```

END AreaOfCircle;

(* ***** Main Program *)
BEGIN
REPEAT
    WriteLn;
    WriteString("Для выбора расчёта введите номер фигуры");
    WriteString("и нажмите ENTER.");
    WriteLn;
    WriteString("Фигуры: 1 - Квадрат, 2 - Прямоугольник, 3 - Треугольник, ");
    WriteString("4 - Круг, 5 - Выход");
    WriteLn;
    WriteString("Ваш выбор : ");
    ReadInt(InChar);

    CASE InChar OF
        1 : AreaOfSquare; |
        2 : AreaOfRectangle; |
        3 : AreaOfTriangle; |
        4 : AreaOfCircle; |
        5 : WriteString("Выход из программы");
            WriteLn;
    ELSE
        WriteInt(InChar, 1);
        WriteString(" Что-то я Вас не понимаю однако... ");
        WriteLn;
    END;
    UNTIL InChar = 5;
END Areas.

```

Ещё одна ошибка будет специфична для тех, кто как и я, в большинстве случаев программирует на Паскале. ☺ Поскольку я люблю шаблоны, для Модулы я тоже сделал шаблон:

```

MODULE pr1;

FROM StrIO IMPORT WriteString, WriteLn;
FROM FpuIO IMPORT WriteReal, ReadReal;
FROM NumberIO IMPORT ReadInt, WriteInt;

BEGIN

    WriteLn("Это конец...");
    WriteLn;

END pr1.

```

Делая очередной пример, я его скопировал с именем файла «pr2.mod»:

```

MODULE pr1;

FROM StrIO IMPORT WriteString, WriteLn;
FROM FpuIO IMPORT WriteReal, ReadReal;
FROM NumberIO IMPORT ReadInt, WriteInt;

FROM MathLib0 IMPORT sqrt;

```

```

VAR
  r1: REAL;

BEGIN
  r1:=sqrt(100.0);
  WriteReal(r1, 10, 2);

  WriteString("Это конец... ");
  WriteLn;
END pr1.

```

Программа благополучно скомпилировалась и при запуске ничего, кроме «Это конец...» не выдавала в принципе. Остатки волос на моей лысине встали дыбом — ошибок нет, по крайней мере синтаксических, а результата тоже нет. Через некоторое время до меня всё-таки дошло, что несмотря на то, что я компилирую «pr2.mod» у меня компилируется отчего то шаблон. После изменения названия модуля во второй программе (с «pr1» на «pr2»), она стала выдавать то, что нужно. ☺

На разборки проблем с Модулой у меня ушло примерно 1 час. Когда я впервые осваивал язык Си, то на сишиные проблемы времени ушло неизмеримо больше. Отсюда вывод — отличие Модула-2 от Паскаль не такие уж и серьёзные... ☺

Выше был Linux, а теперь попробуем Модулу в Windows. Для винды готовых сборок «gm2» я не нашёл, поэтому придётся собирать из исходников [35]. Первые смутные подозрения у меня возникли во время скачивания архива — он показался излишне большим. Распаковав его я понял, что сомнения были не напрасны, это оказался набор компиляторов GCC целиком — C\C++, GNAT, Fortran, ObjC и т.д. Только в отличие от стандартного набора исходников, туда подкинули ещё два каталога — сам компилятор gm2 и библиотеки поддержки. Решив быть самым хитрым в округе, я сначала хотел было собрать только исходники из этих двух каталогов. Но скрипт конфигуратора пожаловался на отсутствие библиотеки libthread. В моей среде MSYS2 отдельно такой библиотеки не нашлось. Точнее нашлась уже установленная версия с расширением «.a», но DLL-ки никакой не было. Откровенно я не сильно пытался выяснить, где бы она могла прятаться по ещё не установленным пакетам, тем более что интернет мне сказал, она по любому должна уже быть установлена. Пришлось прибегнуть к нелюбимому всеми способу разрешения проблем — чтению документации. ☺ В документации был упомянут только CygWin и сборка должна происходить так:

```

/gcc-versionno/configure \
--prefix=/gm2/opt \
--disable-multilib --enable-checking=all \
--enable-languages=c,c++,gm2
$ make

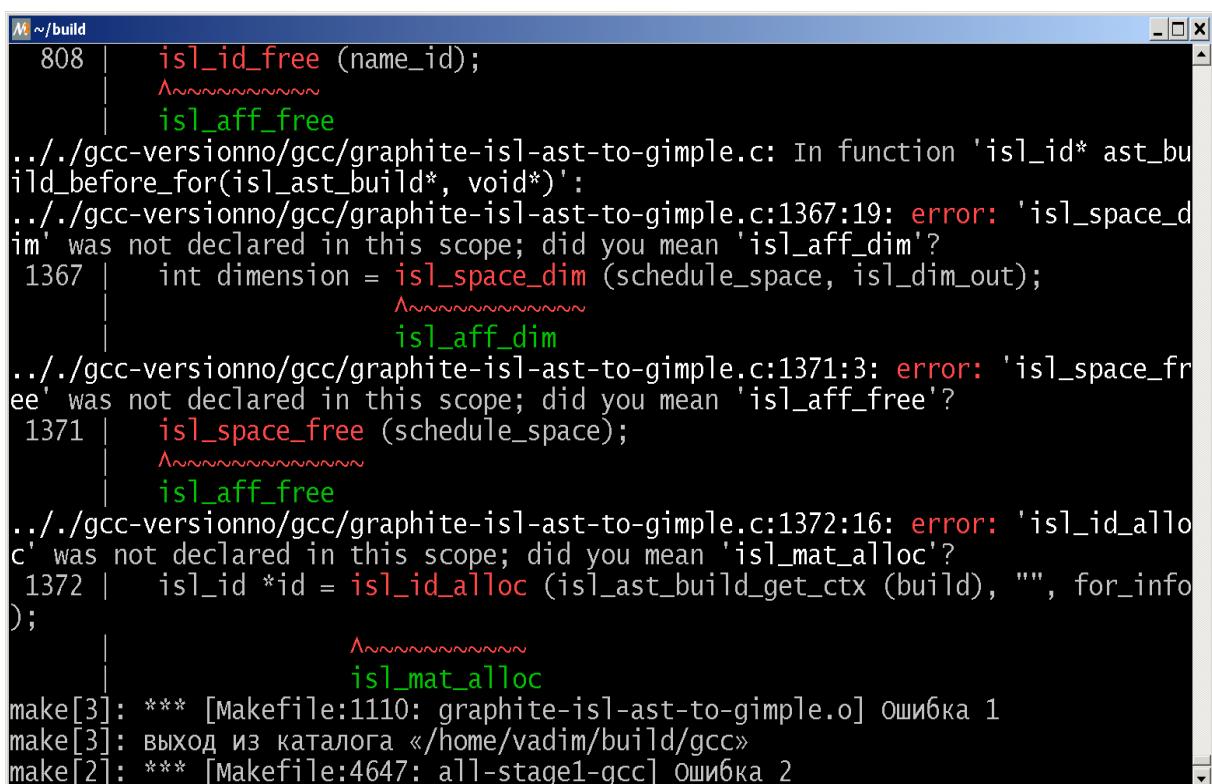
```

Опции конфигуратора в принципе понятны. Ключ «--prefix» интересен

только тем, кто работает исключительно внутри среды CYGWIN или MSYS2. Остальным можно этот ключ игнорировать. Ключ «--enable-checking» — на ваше усмотрение. Полная проверка пока что никому не мешала. ☺ Решил сэкономить время сборки путём исключения оттуда C\С++, строку конфигурации я написал так:

```
/gcc-versionno/configure --enable-languages=gm2
```

Увы, конфигуратор, немного пошуршав в консоли, выдал мне «ая-яй!». Дескать без C\С++ обойтись никак нельзя, поэтому он их сам включил в список целей. Ну что ж, ничего не поделаешь, раз там за меня всё лучше знают, придётся покориться неизбежности. По всей видимости, тут он собирает сначала C\С++, а потом с помощью него собирает gm2. В общем, сборка длилась очень-очень-очень долго, я себе все ногти до самых локтей изгрыз и в результате увидел вот что:



```
~/build
808 |     isl_id_free (name_id);
|     ^~~~~~
|     isl_aff_free
.../gcc-versionno/gcc/graphite-isl-ast-to-gimple.c: In function 'isl_id* ast_build_before_for(isl_ast_build*, void*)':
.../gcc-versionno/gcc/graphite-isl-ast-to-gimple.c:1367:19: error: 'isl_space_dim' was not declared in this scope; did you mean 'isl_aff_dim'?
1367 |     int dimension = isl_space_dim (schedule_space, isl_dim_out);
|     ^~~~~~
|     isl_aff_dim
.../gcc-versionno/gcc/graphite-isl-ast-to-gimple.c:1371:3: error: 'isl_space_free' was not declared in this scope; did you mean 'isl_aff_free'?
1371 |     isl_space_free (schedule_space);
|     ^~~~~~
|     isl_aff_free
.../gcc-versionno/gcc/graphite-isl-ast-to-gimple.c:1372:16: error: 'isl_id_alloc' was not declared in this scope; did you mean 'isl_mat_alloc'?
1372 |     isl_id *id = isl_id_alloc (isl_ast_build_get_ctx (build), "", for_info);
|     ^~~~~~
|     isl_mat_alloc
make[3]: *** [Makefile:1110: graphite-isl-ast-to-gimple.o] ошибка 1
make[3]: выход из каталога «/home/vadim/build/gcc»
make[2]: *** [Makefile:4647: all-stage1-gcc] ошибка 2
```

Рисунок 67. Попытка собрать gm2 в Windows

Непосредственно до gm2 мы даже и не добрались, застряли где-то в общих с Си файлах. Что-то они там не поделили с целочисленной библиотекой ISL. Поиски в интернете показали, что не только у меня и не только с gm2 есть эта ошибка, но как её решить — так и осталось неясным. Один из разработчиков «gm2» написал в новостях, что вынужден отказался от сборки «gm2» совместно с CYGWIN, видимо по той же причине. Официальная версия — слишком далеко «gm2» отошла от Windows. И поскольку сборка Модула-2 в

Windows для меня представляла чисто академический интерес, на этом и закончим... ☺

Поклонникам Windows не стоит огорчаться. Здесь будет прекрасно работать универсальный компилятор для Модула-2/Оберон-2 от российских разработчиков Excelsior под названием «xds» [38]. Правда он только 32-ухразрядный, но для Windows это несущественный фактор. Какой именно язык использовался для написания программы компилятор опознаёт по расширению файла:

- myprog.mod — это будет Модула-2;
- myprog.ob2 — это будет Оберон-2.

Компиляция консольным компилятором делается так:

```
xc =m myprog.mod
```

Здесь ключ =m означает «make». Ключ можно писать и целым словом «=make». Если у вас всего один программный файл, то компилятор создаст исполняемый файл с тем же именем и расширением «exe».

В качестве вывода. Стоит ли сегодня браться за язык Модула-2? Если это предписано вашей работой — тогда стоит. По освоению он немного легче Паскаля, поскольку, в отличие от Паскаля, живёт по очень строгому уставу, образца прусской армии. Там у них не забалуешь, но зато всегда понятно что делать. Синтаксис на паскалевский очень похож. Ключевые слова все те же, правила применения тоже. Единственная проблема — другие названия стандартных модулей и, как следствие, неизвестное распределение процедур по этим модулям. Так что если вам нужно по работе — беритесь смело, проблем будет минимум.

Где реально используется — НПО им. Решетнёва для спутников связи. Есть ли где-то ещё — мне неизвестно. В связи с местом использования у нас наклёвывается совершенно неутешительный вывод: что-либо узнать об эффективности реального применения Модулы-2, а так же возможности посмотреть на реальные коды — нет никакой возможности.

Если же вы захотите побаловаться Модулой просто для себя, для интереса, приготовьтесь к тому, что кроме виртовских книг ничего серьёзного в интернете не найдёте. Есть некоторое количество Модул на Github'e, но это количество не сравнить даже с Паскалем. На мой взгляд лучше подсесть на другой язык, который несколько попроще и поинтереснее — Оберон... ☺

Внуки Паскаля — Oberon, Component Pascal, BlackBox и прочее, прочее, прочее...

Oberon. Это самый последний виртовский потомок Паскаля, возможно недоделанный, потому-что по швейцарскому законодательству как раз на Обероне Вирта выгнали на пенсию как госслужащего (а ETH — государственное учреждение, поэтому мнение Вирта о пенсии никто не спрашивал). Как проект он переселился на персональную страничку Вирта [44] и всё самые свежие нововведения от Вирта лучше смотреть там, потому что Oberon Microsystems [45], которая собственно раньше и была Обероном, стала как-то уж очень подозрительно чиста душою. На ихней страничке нынче мало что можно найти. Похоже Обероном после Вирта они не занимаются, подсели на чисто коммерческие заказы. Краткая суммирующая страница по Оберону здесь [47].

Язык Оберон по популярности в мире сейчас стоит примерно на том же месте, что и Модула-2, т.е. все без исключения о нём знают, многие тырят оттуда идеи для своих разработок, но именно Оберон используют только те, кто достаточно хорошо понимает его неслабые возможности.

Надо сказать, что Оберон с одной стороны очень похож на Модулу, но с другой стороны — это уже принципиально другая ступень работы программ. Виртовский Оберон в первоначальной задумке — это ещё более тесная интеграция с ОС, когда на программиста ложится только необходимость реализовать алгоритм, а прочими обслуживающими вещами занимается ОС, причём сама по себе и без напоминаний и понуканий программиста. Например, и запуском программ занимается ОС и сборкой мусора после работы программы тоже занимается ОС. Все программы работают в едином адресном пространстве и различаются они, если говорить сегодняшними терминами, как потоки у единой программы. Т.е. ОС — это программа запускающая потоки. Таким образом была сделана попытка создания «безопасной среды», когда ОС очень строго следит за всем, что происходит в её недрах. Однако современные ОС общего назначения пошли другим путём — для каждой запущенной программы выделяя собственное адресное пространство. Единственной известной мне попыткой внедрить виртовскую идею одноадресного пространства в промышленных масштабах сделали наши разработчики для суперкомпьютера «Эльбрус». Следжение за программами там осуществлялось не только на уровне ОС, но и аппаратными способами. Был даже создан специальный транслятор для эльбрусовских программ — «Эль-76». Это оказалось делом слишком дорогим, поэтому такая система помимо «Эльбруса» нигде больше не использовалась, да и Эльбрусов сделано не так уж много, потому что для него нужна собственная электростанция, озеро Байкал для охлаждения и не меньше одного полка солдат для технического обслуживания. 😊

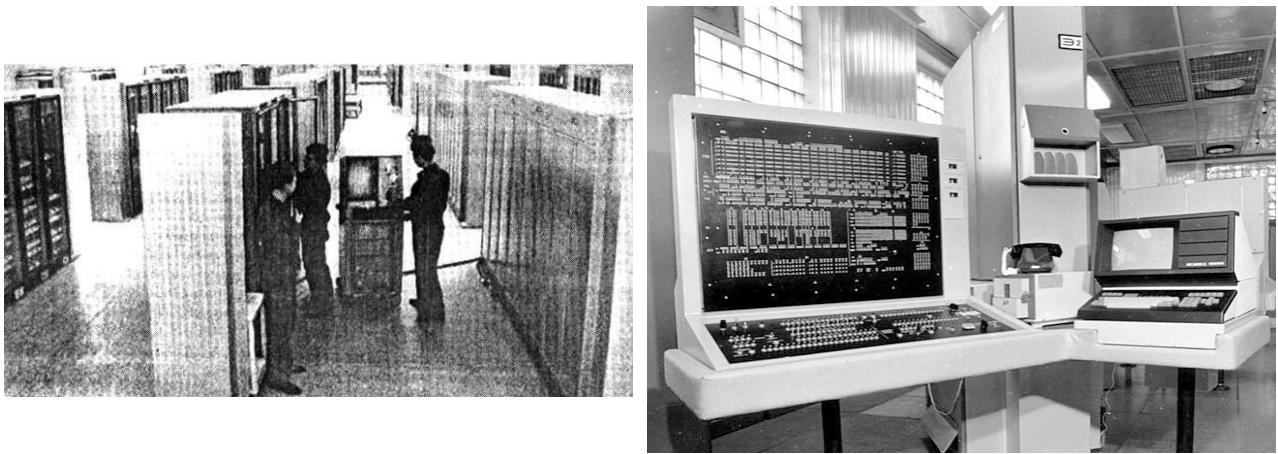


Рисунок 68. СуперЭВМ «Эльбрус-2» и пульт управления.
Вычислительный центр КВЦ 5К80 системы ПРО А-135

Эта концепция сейчас перекочевала в BlackBox [46], которая по сути является виртуальной машиной, работающей поверх какой-либо ОС, скромно прикидываясь обычной программой.

Несмотря на то, что сама концепция Оберона просто чудо как хороша, но за ней, как и в случае Модулы, не стояла и не стоит никакая могучая финансовая структура, типа как за Microsoft, поэтому никто бешеной рекламной компании для него не проводил. Однако, на мой взгляд, малое распространение Оберона связано не только с этим. На сегодняшний день все ОС общего назначения придерживаются другой концепции «безопасной среды» — каждая программа сама по себе и поэтому ей выделяется собственное адресное пространство для функционирования. Фактически ОС создаёт для каждой запускаемой программы виртуальную машину, в которой та и работает. Т.е. каждая запущенная программа ничего не знает о своих соседях или работающем оборудовании. Из-за этого приходится использовать системные библиотеки, которые эту самую концепцию «моя хата с краю» нарушают. С концепцией Оберана «все люди — братья» это мало согласуется.

На сегодняшний день можно сказать, что Оберон является слегка упрощённой версией Модулы-2. Тексты программ на Обероне читаются очень легко из-за того, что ключевые слова языка выделяются большими буквами. Самых ключевых слов, по сравнению с другими языками, довольно мало. Из принципиальных отличий — исчезли зачатки параллельного программирования. Понятно, что если этими вопросами должна заниматься ОС, то вводить параллелизм в язык программирования — явное излишество. Однако можно подключить системную библиотеку, сделать интерфейсный модуль и реализовать многопоточное программирование с помощью него, как это сделано в Delphi и FreePascal на основе класса TThread.

Претерпела изменение концепция модулей. В Обероне модуль — это не два отдельных файла (один объявление и один — реализация), а один. Признак

экспортирования объектам добавлялся в виде специальной закорючки к имени объекта — «*» если объект для чтения и записи, и «-» если объект только для чтения.

В первоначальной версии Оберона убрали даже цикл FOR (в Фортране, кстати, его тоже как такового нет, но есть модификация основного бесконечного цикла), объясняя это тем, что FOR является частным случаем WHILE. Однако в Оберон-2 его опять вернули, поскольку он даёт более наглядный код при работе с массивами.

ООП (точнее, то что мы можем признать за ООП 😊) опирается на те же принципы, что и Модула-2, т.е. в пределах модуля все всё видят, а за пределами — только то, что экспортируется (или чтение\запись или только чтение). Кстати сам экспорт представлен в более компактном виде. Процедуры, которые обрабатывают данные, можно считать изначально виртуальными, поскольку их переопределение осуществляется без добавления специальных языковых примочек, как это практикуется в нынешнем Паскале\Delphi. Таким образом модуль является примерным аналогом класса в ООП (не дельфийского, а сиплюсплюснутого), у которого обмен данными между модулями-классами осуществляют экспортируемые модулем процедуры. С наследованием — вопрос скользкий. Дело в том, что у классического ООП наследование довольно часто происходит без знания полной структуры базового класса, что порою делает проблематичным модификацию класса. Да, можно посмотреть исходник класса, но в случае с C++, его просто можно не понять или не всё понять. Да и создателей наследников интересуют только интерфейсные компоненты, а не скрытые. В общем, объекты в виртовском Обероне остались в виде MODULE, а класс — в виде указателя на RECORD.

ООПовское наследование можно сделать для типов RECORD. К примеру, есть у нас некий объект типа «точка»:

```
MyPoint* = RECORD
    x*: INTEGER;
    y*: INTEGER;
    Draw*: PROCEDURE();
END
```

На его основе можно создать тип «окружность»:

```
MyCyrcle* = RECORD (MyPoint)
    radius*: INTEGER;
    Draw*: PROCEDURE();
END
```

Здесь, после слова RECORD, мы в скобках помещаем название объекта, на основе которого делается новый объект. В дополнение к существующим полям базового объекта (координатам точки), у нас появилось дополнительное поле «radius» и переопределена процедура рисования. Переопределение работает без каких-либо синтаксических прибамбасов, должны совпадать

только имя процедуры-метода и типы параметров. Если переопределения никакого не будет, то в наследуемом типе действует процедура базового типа.

Некоторое время обсуждался вопрос введения беззнаковых типов, однако вводить не стали, т.к. это усложняет обработку чисел. В Фортране, кстати говоря, беззнаковые числа тоже до сих пор не ввели, хотя для стандарта Фортран-202x эта возможность в комитете по стандартизации обсуждается. Надо сказать, что беззнаковые типы вводились вопреки математике, где таких типов просто нет. Вызвано это было тем, что компьютеры имели довольно убогие ресурсы и таким хитрым образом хотели расширить числовой диапазон обрабатываемых значений. Это сильно усложнило и производство трансляторов и повысило вероятность ошибки в программе.

Классический Оберон, в его доработанной версии (Оберон-2), можно изучать с помощью книжки Вирта и Мессенбека (тоже создателя Оберон-2) [48]. Насколько хорош Оберон в объектной ориентации, можно почитать в книге [53]. Международного стандарта, в отличие от Модула-2, у Оберона нет и, наверное, уже не будет. На сегодняшний день Оберон довольно широко применяется для программирования микроконтроллеров. Еще при Вирте (он жив, не падайте в обморок 😊, имеется в виду, когда тот работал в ETH 😊) был создан беспилотный вертолёт, бортовое ПО которого писалось на Обероне.

В отличие от Модулы, которая фактически существует только в виде Модула-2, Оберонов сейчас множество. Это можно считать проблемой, потому что приходится тщательно исследовать каждый компилятор-транслятор-среду разработки на предмет, а как же она реально работает. Все основные операции обычно работают одинаково и основные модули (опять же, в отличие от Модулы 😊) одинаково называются, но есть нюансы...

Основные версии Оберона:

- **Оберон.** Это самая первая экспериментальная площадка Вирта, реальных её воплощений нет ни у кого. Однако он идёт, как пример, к книге Вирта «Построение компиляторов»;
- **Оберон-2.** Первый реальный и практический релиз Оберона, 16-тиразрядный. Сейчас именно 16-тиразрядную версию наверное уже нигде не встретишь, однако Оберон-2 часто предстаёт уже в более осовремененном 32/64-хразрядном виде, в который добавлено кое-что из следующего виртовского стандарта;
- **Оберон-07 или Оберон-2013.** Два названия, потому что основу идеи модификации Оберона Вирт подготовил к 2007 году и у него на сайте он носит название Oberon-07, но закончилась эта модификация только в 2013 году. Примерно тогда же вышла специальная книжка Вирта и другого разработчика, Юрга Гуткнхта, «Project OberonThe Design of an Operating System, a Compiler, and a Computer. Revised Edition 2013». К сожалению пока что не все разработчики переключились на этот стандарт, видимо считая, что и Оберон-2 неплох;

- **Компонентный Паскаль** или, как его ещё называют по среде разработки — **BlackBox**. Это очень сильно модифицированный Оберон уже не виртовской разработки, цель которого приспособить Оберон к объектному и визуальному программированию путём подключения сторонних компонентов. В России BlackBox в основном используется как образовательная платформа для обучения программированию, так же как PascalABC.NET. Менее заметная роль — программирование микроконтроллеров;
- **Active Oberon** и его .NET реализация **Zonnon**. Это параллельная BlackBox'у ветка Оберона, так же затачиваемая под ООП, плюс встроенная в объект многопоточность. Отчасти дублирует Компонентный Паскаль по своим возможностям, поскольку позволяет встраивать в программу сторонние компоненты, которые открыты операционной системе для использования (аналог Microsoft COM). Отдельные российские ВУЗы делали попытку ввести его в образовательную программу, но насколько это было успешно, а так же о наличие каких-то успешных бизнес-проектов на основе этого языка, мне неизвестно.

Если говорить про «стандарт», на который следует ориентироваться при выборе компилятора\транслятора, то необходимо уточнять в документации, что релиз поддерживает либо наиболее поздний виртовский стандарт Oberon-07(Oberon-2013) [61], либо Оберон-2. И желательно чтобы транслятор был позднее 2013 года.

Вот основные отличия Оберон-07 от Оберон-2 (и более старой Модулой):

- Ключевое слово RETURN (возвращаемое_что-то) у PROCEDURE теперь должно стоять только перед самым END Имя_Процедуры, фактически являясь синтаксической частью END. По объяснениям Вирта это сделано для того, чтобы не допустить неправильного результата в случае, если результат будет формироваться где-то не в конце процедуры-функции, а, например, с помощью CASE и таким образом программист будет видеть, что именно он отдаёт. Если необходим какой-то многовариантный разветвляющийся RETURN, то это можно организовать путём использования дополнительной переменной.

```
PROCEDURE MyMin(a, b : INTEGER) : INTEGER;
VAR mmin : INTEGER;
BEGIN
  IF a < b THEN
    mmin := a
  ELSE
    mmin := b;
  END
  RETURN mmin END MyMin;
```

- Удалено, как явление, служебное слово WITH, которое создавало неслабую возможность перепутать одинаковые поля у разных объектов.
- Поскольку ввели единообразие в нумерацию массивов (начинается с 0), заодно решено было упростить обмен данными между переменными с однотипными массивами и переменными с однотипными RECORD'ами. Вместо каких-нибудь заумных процедур теперь можно делать обычное присваивание.

```
VAR
  a: ARRAY 10 OF INTEGER;
  b: ARRAY 20 OF INTEGER;
BEGIN
  ...
  b := a;
  ...
END;
```

Естественно, принимающий массив должен быть не короче передающего, длиннее — это пожалуйста. ☺

- Сложные типы RECORD у нового Оберона в процедуру теперь передаются сложно. ☺ Видимо чтобы не копировать значения, они передаются по ссылке. Это не эквивалентно VAR, поскольку не вернёт в основную программу сделанные изменения. Этот способ просто экономит время, как в Фортране.
- Удалены типы SHORTINT и LONGINT. Первое — потому что для увеличения скорости обработки всё равно требуется выравнивание данных по размеру машинного слова. Второе потеряло смысл, потому что у всех нынешних процессоров SIZE(LONGINT) = SIZE(INTEGER). Теперь размер INTEGER соответствует размеру машинного слова — в 32-ухразрядных системах равен 4 байтам, в 64-ёхразрядных — 8 байтам.
- Цикл WHILE усложнили дополнительной возможной секцией ELSE. Я, честно говоря, так и не смог придумать какого-нибудь примера, где без ELSE никак не обойтись. ☺ Сам Вирт объясняет такую конструкцию влиянием своего друга Дейкстры. Выглядит это так:

```
WHILE a>b DO
  a := a - 1
ELSE b>a DO
  b := b - 1
END
```

Т.е. получается комбинация IF+WHILE. Цикл теперь, как IF, может быть с двумя ветками. Выполняется та из них, которая соответствует заданному условию.

- LOOP, который портил скорострельность циклов необходимостью проверки дополнительных условий внутри цикла, удалён за

ненадобностью.

- Аварийный выход из программы осуществляется процедурой ASSERT(логическое_выражение). Если «логическое_выражение» равно FALSE, то программа завершается.

Несколько общих вещей для всех Оберонов.

По сравнению с Паскаль в Обероне изменилась концепция «безразмерных массивов» — ARRAY OF какой-то_тип. Такой массив можно передать в качестве параметра в процедуру, а вот объявить самостоятельную переменную такого типа не удастся. Необходимо будет объявить указатель на массив с каким-нибудь огромным количеством ячеек, а в программе уже выделить память под конкретное количество:

```
VAR
  a: POINTER TO ARRAY 1000000 OF INTEGER;

BEGIN
  ...
  NEW(a, 478);
  ...
END.
```

Оберон позволяет работать в программе с достаточно большими статическими массивами без опасения вызвать STACK OVERFLOW, что делает необходимость использования «безразмерных массивов» не таким уж и частым.

В отличие от Pascal или Delphi, модуль SYSTEM к вашей программе автоматически не подключается. Мало того, файла с обероновским кодом с таким названием вы не найдёте. Это виртуальный модуль. Ничего странного тут нет. В отличие от Pascal\Delphi в модуле SYSTEM процедур очень мало. Что именно — можно посмотреть в [60, 61]. Например, можно получить адрес какой-то процедуры или переменной. Можно проверить какой-нибудь бит в целочисленном значении. В каких-то конкретных трансляторах разработчики могут включить в него дополнительные процедуры. Но в любом случае, не исключено, что вашей программе эти процедуры вовсе не понадобятся и подключать его нет смысла.

Однако, если вы подключаете какой-то стандартный модуль, например «Out» для вывода значений в консоль, то там внутри него модуль SYSTEM конечно же будет использоваться, однако для вас недоступен. Доступно только то, что вы подключаете явно, своими ручками.

Оберон и системные библиотеки. В общем случае правило их подключения соответствует Pascal\Delphi. Если транслятор поддерживает статическое связывание, то всё ограничивается заголовочным объявлением процедуры. Если же только динамическое связывание, то вы сначала

открываете какой-то файл динамической библиотеки, получая на неё дескриптор, а потом по этому дескриптору вытаскиваете адрес нужной вам функции.

Оберон и ассемблер. Именно ассемблера, т. е. мнемонических команд процессора, я пока что ни у кого не видел. И в виртовском стандарте такого вообще нет. Однако у некоторых трансляторов, в частности «akron», можно выполнять шестнадцатиричные команды процессора с помощью SYSTEM.CODE(какие-то нужные команды). Тут уж нужно смотреть возможности каждого конкретного транслятора.

Не во всех дистрибутивах Оберона я заметил побитовые операции над целыми числами (может быть плохо глядел \odot) — and, or, xor, not. Это можно поправить с помощью преобразования целых чисел в множества (SET), потом проведения с SETами операций пересечения, сложения, разности и отрицания. Выглядеть это будет так (кстати, хорошая иллюстрация, зачем вообще нужны множества \odot):

```
(* Побитовые операции над целыми числами *)
MODULE MathBits;

PROCEDURE iand* (x: INTEGER; y: INTEGER): INTEGER;
BEGIN
    RETURN ORD(BITS(x) * BITS(y))
END iand;

PROCEDURE ior* (x: INTEGER; y: INTEGER): INTEGER;
BEGIN
    RETURN ORD(BITS(x) + BITS(y))
END ior;

PROCEDURE ixor* (x: INTEGER; y: INTEGER): INTEGER;
BEGIN
    RETURN ORD(BITS(x) / BITS(y))
END ixor;

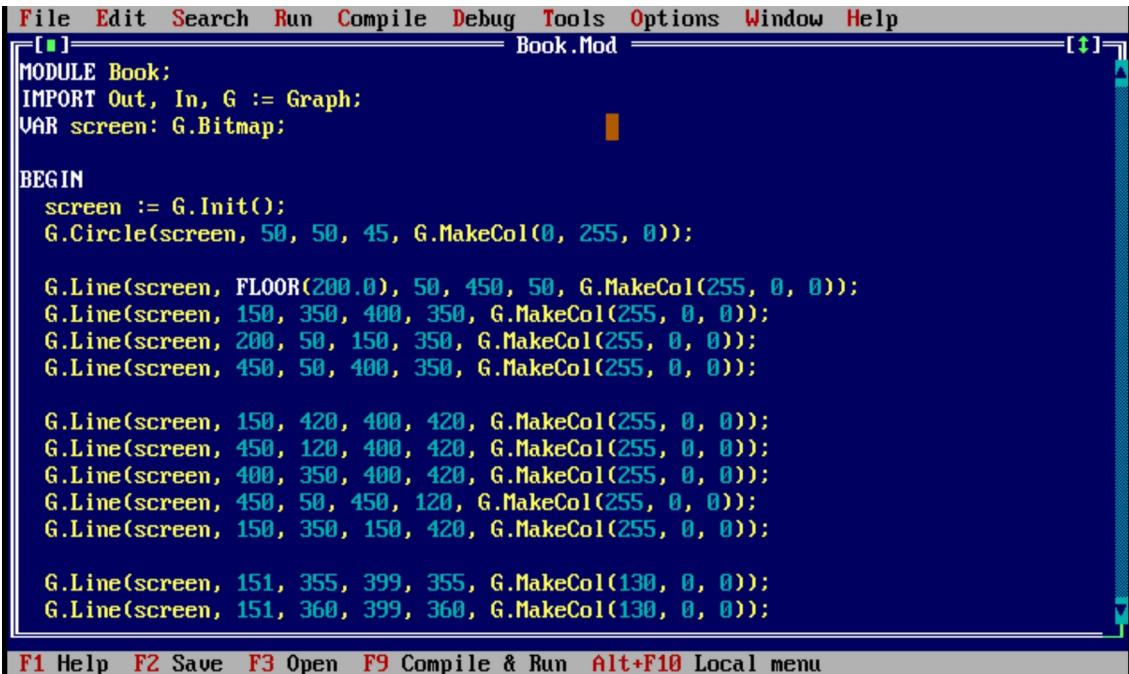
PROCEDURE inot* (x: INTEGER): INTEGER;
BEGIN
    RETURN ORD(-BITS(x))
END inot;

END MathBits.
```

Дистрибутивы Оберона

Наверное самым лёгким дистрибутивом для первого знакомства и для «поиграться» можно назвать FreeOberon [49], который является российской разработкой и выглядит весьма ностальгичненько... ☺ Предполагалось, что он будет использоваться в качестве замены «застрелиться_и_не_встать» TurboPascal'я на Оберон-2 в учебных заведениях. Есть версия для Windows и версия для Linux. Возможно из исходников его можно скомпилировать и для FreeBSD. Какими либо страшными зависимостями не страдает и в нём можно начинать работать сразу после установки. Как учебный полигон подходит прекрасно.

FreeOberon в версии для Windows имеет на борту уже откомпилированную программную оболочку, компилятор Оберона и необходимые библиотеки (в частности SDL2 для работы графики). Таким образом его можно просто распаковать и запускать в работу. С Linux чуть посложнее. В архиве идёт только исходник. Прежде чем его собирать, необходимо дополнительно установить из репозитория вашей ОС пакеты «libsdl2-dev» и «libsdl2-image-dev», причём именно с приставкой «-dev» (или «-devel», это уже зависит от ОС), т.е. версию для разработчика. Обычно сами библиотеки уже установлены, однако Оберон, как и FreePascal с Lazarus, в упор не желает замечать библиотеку, если у неё после «.so» есть ещё какой-нибудь цифровой хвост [50]. После этого исходники прекрасно компилируются и для Linux 32 и для Linix 64. В главном каталоге появляется исполняемый файл «FreeOberon», который можно немедленно запускать.



The screenshot shows the FreeOberon IDE interface. The menu bar includes File, Edit, Search, Run, Compile, Debug, Tools, Options, Window, Help, and a local menu. The title bar displays 'Book.Mod'. The code editor contains the following Pascal code:

```
MODULE Book;
IMPORT Out, In, G := Graph;
VAR screen: G.Bitmap;

BEGIN
  screen := G.Init();
  G.Circle(screen, 50, 50, 45, G.MakeCol(0, 255, 0));
  G.Line(screen, FLOOR(200.0), 50, 450, 50, G.MakeCol(255, 0, 0));
  G.Line(screen, 150, 350, 400, 350, G.MakeCol(255, 0, 0));
  G.Line(screen, 200, 50, 150, 350, G.MakeCol(255, 0, 0));
  G.Line(screen, 450, 50, 400, 350, G.MakeCol(255, 0, 0));
  G.Line(screen, 150, 420, 400, 420, G.MakeCol(255, 0, 0));
  G.Line(screen, 450, 120, 400, 420, G.MakeCol(255, 0, 0));
  G.Line(screen, 400, 350, 400, 420, G.MakeCol(255, 0, 0));
  G.Line(screen, 450, 50, 450, 120, G.MakeCol(255, 0, 0));
  G.Line(screen, 150, 350, 150, 420, G.MakeCol(255, 0, 0));
  G.Line(screen, 151, 355, 399, 355, G.MakeCol(130, 0, 0));
  G.Line(screen, 151, 360, 399, 360, G.MakeCol(130, 0, 0));
```

The status bar at the bottom shows keyboard shortcuts: F1 Help, F2 Save, F3 Open, F9 Compile & Run, Alt+F10 Local menu.

Рисунок 69. Российский FreeOberon

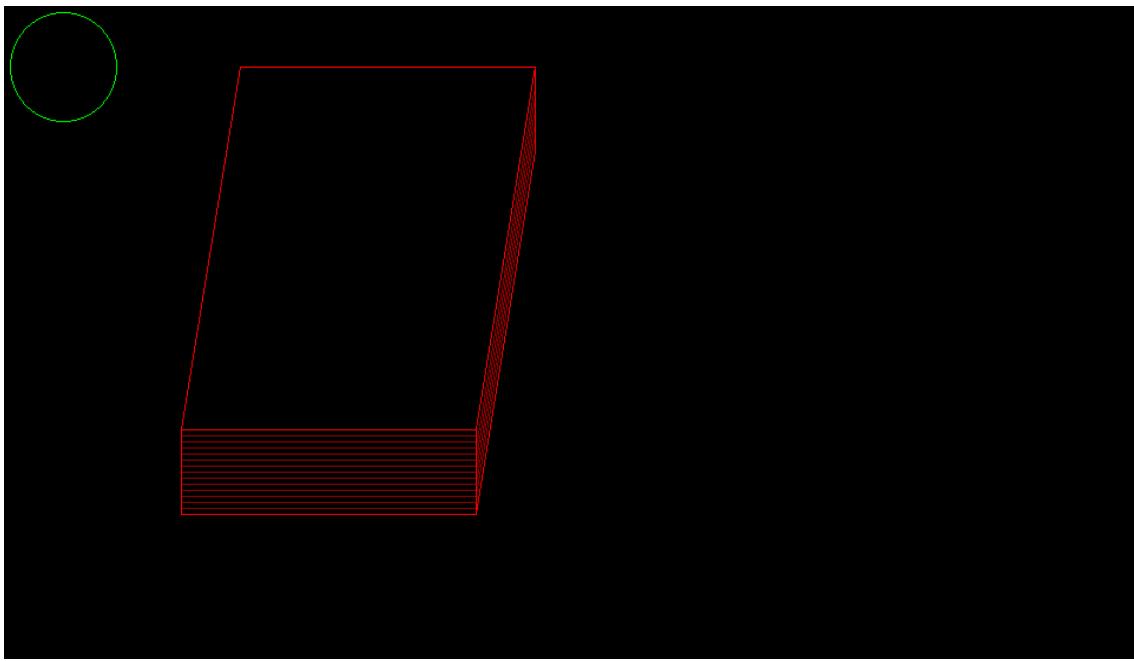


Рисунок 70. Результат запуска программы из предыдущего окошка

Серьёзный недостаток FreeOberon — отсутствие удобного способа мышкой (или не мышкой) открывать файлы с текстами программ. Окошко «Open File...» у них явно ещё не доделано и пользоваться им неудобно.

FreeOberon работает на основе слегка доработанного консольного компилятора «voc» (Vishap Oberon Compiler, фронт-энд для GCC. Vishap — по-армянски «дракон» ☺), который закопан довольно глубоко:

```
каталог_установки/data/bin/voc
```

Компиляция производится через специальные скрипты, которые лежат в «каталог_установки/data/bin/».

Нам тоже никто не мешает делать то же самое, если IDE'шка нас не устраивает. Правда в этом случае логичнее было бы скачать оригинальный «voc» и им пользоваться. ☺

В качестве вишенки на торте, в комплекте с FreeOberon дополнительно идёт среда выполнения программ из BlackBox. В отличие от оригинала, эта среда не графическая, а чисто консольная. Правда, как и в оригиналe, 32-ухразрядная, так что для работы должен быть 32-ухразрядный комплект библиотек, если ваша ОС 64-ёхразрядная.

Оригинальный компилятор «voc» разработан на основе стандарта Оберон-2 и входит в состав некоторых ОС *NIX, например в FreeBSD. Его можно изготовить и самостоятельно, скачав исходники с Гитхаба [51]. Поскольку это

не самостоятельный инструмент, а всего лишь фронт-энд, у вас должен быть доступен какой-либо си似的 компилятор: gcc, clang или msc. Последнее — естественно только под Windows.

В Linux «voc» по умолчанию устанавливается в каталог «/opt/voc», поэтому вам придётся добавить этот путь в переменную PATH. Аналогично и в Windows, но там уж вы каталог установки будете выбирать на свой вкус, опять же не забыв дополнить переменную PATH. Компилятор собирается без малейших проблем, при условии что вы, так же как и в случае с FreeOberon, не забыли про libsdl2 и libsdl2-image.

Компиляция обероновских программ из командной строки в простейшем случае будет происходить так:

```
voc MyProg.ob -m
```

Ключ «-м» означает, что «MyProg.ob» — главный модуль. После компиляции можно заметить промежуточный выхлоп от «voc» — файл «MyProg.c». Это результат преобразования, который потом компилируется в бинарный файл.

Теперь проверим, как работает «voc» на FreeBSD. На Гитхабе написано, что есть уже готовый бинарный пакет в репозитории. Так и оказалось, пакет даже был свежим. К нему в комплекте так же будет добавлен «sdl2» и «sdl2-image». GCC у меня не установлен, зато есть «clang», который в FreeBSD идёт по умолчанию, как системный. Из рисунка ниже видно, что «voc» это нисколько не расстроило, с ним он тоже прекрасно работает:

```
root@srv-clientctrl:~/projects/obr# gcc --version
gcc: Команда не найдена.
root@srv-clientctrl:~/projects/obr# clang --version
FreeBSD clang version 8.0.1 (tags/RELEASE_801/final 366581) (based on LLVM 8.0.1
)
Target: x86_64-unknown-freebsd12.1
Thread model: posix
InstalledDir: /usr/bin
root@srv-clientctrl:~/projects/obr# voc p1.ob7 -m
p1.ob7 Compiling p1. Main program. 379 chars.
root@srv-clientctrl:~/projects/obr# ./p1
Hello, gays!
root@srv-clientctrl:~/projects/obr#
```

Рисунок 71. Компилятор «voc» на FreeBSD

Оберон несколько более демократичен, чем Модула. Например, дополнительный модуль у него подключается целиком, выдирать оттуда отдельные процедуры не нужно. Некоторое уныние вызывает обязательное полномасштабное упоминание названий процедур в тексте программы, т.е.

«Модуль.Процедура». Однако есть возможность подсократить синтаксис, используя вместо полного наименования псевдонимы. И не стоит забывать, что Оберон — это уже почти ООП. Только объект у него не в виде переменной класса, как в ObjectPascal, а в виде модуля, поэтому упоминание объекта — модуля-хозяина, обязательно, чтобы не было путаницы с одноименными методами/переменными из разных модулей.

Ввод\вывод для разных типов помещён в два модуля: In и Out. Ввод или вывод того или иного типа обязательно уточняется с помощью модификатора, например:

```
MODULE MyProg;
IMPORT In, Out;
VAR a, b, c: INTEGER;

BEGIN
  Out.String('Введите первое слагаемое: ');
  In.Int(a);
  Out.String('введите второе слагаемое: ');
  In.Int(b);
  c := a + b;
  Out.String('Сумма равна: ');
  Out.Int(c, 0);
  Out.Ln;
END MyProg.
```

Как и в Модуле, если нужно сделать перенос строки после вывода, для этого нужна отдельная катафасия — «Out.Ln».

Типы данных в Обероне остались такие же, как в Модуле, однако у Вирта в самых последних редакциях Оберона немножко поменялась концепция максимальных\минимальных значений. Теперь она зависит от разрядности транслятора, т.е. всё стало проще и, одновременно, прямо соответствовать операционке. Типы LONGINT и LONGREAL стали просто не нужны. К большому сожалению не все производители трансляторов, в том числе и «voc», прониклись такой здравой идеей, поэтому когда вы берёте какой-то совершенно определённый транслятор, в первую очередь необходимо смотреть на поддерживаемые им типы данных и какие у этих типов граничные значения. В подавляющем большинстве случаев, INTEGER и REAL будут 32 разряда, а значит присутствуют LONGINT и LONGREAL — 64 разряда, вне зависимости от разрядности ОС/транслятора. Давайте проверим, как с этим обстоят дела у компилятора «voc», состряпав такую программку:

```
MODULE A;
IMPORT Out;

VAR nn : INTEGER;

BEGIN
  Out.String("HELLO");
  Out.Ln;
  nn := SIZE(REAL);
```

```

Out.String("Размер REAL = ");
Out.Int(nn, 0);
Out.Ln;
Out.String("Размер LONGREAL = ");
nn := SIZE(LONGREAL);
Out.Int(nn, 0);
Out.Ln;
END A.

```

И вот что получаем в результате:

```

vadim@mycomp1:~/progs/FreeOberon/Programs$ 
voc A.Mod -m
A.Mod  Compiling A. Main program. 640 chars.
vadim@mycomp1:~/progs/FreeOberon/Programs$ ./A
HELLO
Размер REAL = 4
Размер LONGREAL = 8

```

Рисунок 72. У «voc» типы данных пока ещё по старому...

Увы и ах, «voc» к последнему Вирту пока что не прислушался...

Большой интерес представляет компилятор Антона Кротова [52], который называется «akron». Он кроссплатформенный (Win32\64, Linux32\64, KolibriOS), а так же микроконтроллеры MSP430 и STM32. Он может на одной платформе (например Linux) делать компиляцию программы (в том числе и самого компилятора) для другой платформы или микроконтроллера.

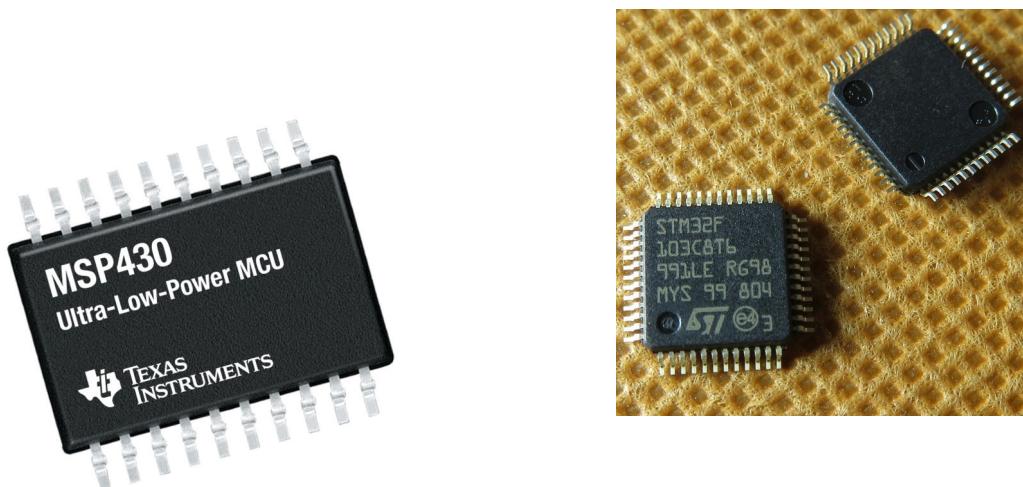


Рисунок 73. Микроконтроллеры поддерживаемые компилятором «akron»

В нём заявлена поддержка стандарта Оберон-07. Проверяем, компилируя 64-ёхразрядным компилятором:

```
ter-master>Compiler.exe size_real.ob07 win64con -out size_real.exe -stk 2
Akon Oberon Compiler v1.26 (64-bit)
Copyright (c) 2018-2020, Anton Krotov
-----
compiling API (SYSTEM)
compiling RTL (SYSTEM)
compiling WINAPI (SYSTEM)
compiling In (SYSTEM)
compiling Out (SYSTEM)
compiling Console (SYSTEM)
compiling size_real (SYSTEM)
-----
1553 lines, 0.05 sec, 8704 bytes
Для продолжения нажмите любую клавишу . . .

D:\....\temp\oberon-07-compiler-master>size_real.exe
Размер типа REAL: 8
```

Рисунок 74. Компилятор Антона Кротова стандарту Оберон-07 соответствует

Как видно, здесь тип REAL — 8 байт, т.е 64-ёхразрядный, а не 32-ух, как это было в «voc». Значит стандарту Оберон-07 он соответствует. Но в бочке мёда отыскалась и ложка дёгтя — отсутствует сборщик мусора и динамическая подгрузка модулей.

Компилятор для Linux разработчик слегка недоделал — скомпилированному бинарному файлу атрибут «выполнимый» отчего-то не присваивается, хотя компиляция проходит успешно. Для исправления этой ситуации, я сделал пакетный файл «oberon», который после компиляции присваивает нужный атрибут:

```
#!/bin/bash

fl=$1
/home/vadim/progs/oberon-07/Compiler $1 linux64exe -stk 2 | chmod 755
chmod 755 ${fl%.ob07}
```

Здесь сам компилятор остался лежать в каталоге, куда я его распаковал (~/progs/oberon-07), а пакетный файл поместил в «/usr/local/bin/», чтобы было видно, что этот файл не из репозитория. Каталог «lib» так же нужно скопировать в «/usr/local/bin/», либо сделать туда симлинк на него. Компиляция программы будет происходить так:

```
oberon myprog.ob07
```

Теперь небольшой примерчик по использованию системных библиотек. В akron отсутствует функция system(), которая довольно просто может запускать

внешние программы. Код извлечения этой функции из библиотеки libc для Linux будет выглядеть примерно так:

```
MODULE call_uname;

IMPORT SYSTEM, Libdl, Out;

VAR
    lib: INTEGER;
    Retval : INTEGER;
    system : PROCEDURE [linux] (cmd: INTEGER): INTEGER;

PROCEDURE System(s: ARRAY OF CHAR): INTEGER;
BEGIN
    RETURN system(SYSTEM.ADR(s))
END System;

BEGIN
    lib := Libdl.open("libc.so.6", Libdl.LAZY);
    SYSTEM.PUT(SYSTEM.ADR(system), Libdl.sym(lib, "system"));
    ASSERT(system # NIL);

    Retval := System ("uname -a");
    IF Retval # 0 THEN
        Out.String("Exit code: ");
        Out.Int(Retval, 0);
        Out.Ln;
    END;

    Retval := Libdl.close(lib);
END call_uname.
```

По сравнению с языком Ада выглядит сложновато. Дело в том, что статическое связывание в akron пока что работает только в Windows, до Linux с этим вопросом разработчик пока не добрался, поэтому придётся использовать динамическое связывание. Для этого нам понадобится модуль Libdl. В сишную функцию нужно передавать строку команды в виде типа PChar. В Обероне такого нет. Обероновская строка похожа на Pchar только наполовину — хоть строка и заканчивается нулём, что позволяет применять большие строки, однако такая строка не является указателем. Из-за этого придётся использовать двухэтажную конструкцию — оригинальная функция из libc с указателем в качестве параметра и надстройка, которая превращает обероновскую строку в указатель. В принципе можно было бы обойтись и без надстройки, но выглядеть будет ещё хуже. ☺ Прототип для системной функции:

```
system : PROCEDURE [linux] (cmd: INTEGER): INTEGER;
```

Типу параметра не удивляйтесь — указатель в Обероне имеет тип INTEGER. В надстройке System() мы передаём в системную функцию адрес строки с помощью SYSTEM.ADR().

Файл библиотеки открываем с помощью уже имеющейся функции open(),

которая является надстройкой системной функции `dlopen()`. Далее по полученному дескриптору файла с помощью функции `sym()`, которая является надстройкой системной функции `dlsym()`, получаем адрес нужной нам функции. В переменную «`system`» адрес перекладываем с помощью `SYSTEM.PUT()`. С помощью процедуры `ASSERT()` проверяем, получен адрес или нет. В принципе той же самой процедурой можно проверять и правильность открытия библиотеки.

Для одной функции такая система выглядит громоздкой, однако если взять систему обмена между вашей программой и базой данных (СУБД), то там речь уже пойдёт о сотне функций и само собой напрашивается необходимость отдельного модуля для этого дела. Если грамотно распределить в этом модуле обязанности между интерфейсными и служебными процедурами, мы получим прямой аналог паскалевского класса `TDataSet` и вся сложность будет спрятана внутри, а работа с БД не будет отличаться от привычной в Delphi или Pascal.

Среда разработки от российской компании Excelsior [38] (с компилятором) заявлена, как поддерживающая два языка — Модула-2 и Оберон-2. И если Модула-2 реализована в полном объёме ISO 10514, о чём заявлено на русскоязычном сайте [37], то в каком объёме реализован Оберон-2, не совсем понятно. Будем надеяться, что тоже в полном виртовском. Эта штуковина может успешно использоваться как универсальное средство программирования для двух языков, не особо заморачиваясь перепрыгиванием в разные IDE'шки и на разные инструменты. Благо что сами языки очень похожи. Заявлено так же, что присутствуют модули поддержки ОС как Windows (WinAPI), так и Linux (POSIX и X11). Заголовки для подключения в программу лежат в подкаталоге «`def`» того каталога, куда вы рапаковали «`xds`». Официально разработка транслятора давно заброшена (примерно с 2009 года). Транслятор для Linux лежит на Sourceforge [36]. Он (как и для Windows) полностью рабочий. Есть только одна проблема — 32-ухразрядность. Для Windows это не проблема, а вот в Linux придётся ставить второй комплект системных библиотек и 32-ухразрядный компилятор GCC, иначе мы получим только промежуточный двоичный файл, но не исполняемую программу. У разработчиков была заявлена IDE'шка в комплекте к `xds` (которая так и называлась — `xds`), но мне найти её не удалось. На Гитхабе есть продвинутая IDE'шка на основе Eclipse [39], но она заточена под Модула-2, под Оберон видимо доделать не успели. Так что если кому очень нужно, можно её попробовать. Кстати говоря, на том же Гитхабе есть несколько проектов `xds`, наверное в одном из них можно будет найти IDE'шку, которая раньше была в комплекте.

Ещё одна особенность — размер INTEGER равен 2 байтам. Если будете использовать этот компилятор не забывайте такую интересную особенность. ☺

```
vadim@mycomp1:~/Документы/www.freepascal.ru/Статьи/История языка Паскаль/code/temp$ wine p3.exe
Sizes of type variables:
INTEGER = 2
LONGINT = 4
REAL = 4
LONGREAL = 8
```

Рисунок 75. Размеры типов данных в «xds»

В архиве есть специальный транслятор из Модулы\Оберона в Си — «xm». С помощью него происходит конвертация кода в чистый K&R С. Таким образом можно программы компилировать на тех платформах, где «xc» не работает. Правда придётся перетащить большой каталог сицких заголовков, в которых лежат макросы, превращающие сицкий псевдокод Модулы\Оберона, в настоящий сицкий.

Компилятор (xc.exe) опознаёт обероновский файл по расширению «ob2». Компиляция будет такая же, как и в случае для Модулы:

```
xc =m myprog.ob2
```

однако в отличие от Модулы, в первой строке вашей программы нужно поставить специальный признак, что это главный модуль, иначе программа откомпилируется, но исполнимый файл не создастся:

```
<*+ MAIN *>
MODULE myprog;

IMPORT Out;

BEGIN
  Out.String("Всем привет!");
  Out.Ln;
END myprog.
```

Ещё один довольно интересный компилятор — «obc» [76], спроектированный не где-нибудь, а в Оксфордском университете. Это транслятор Оберон-2, но с некоторыми дополнениями от Оберон-07 (см. по ссылке [76] «Notes on changes in Oberon07»). Написан на языке «Objective Caml», а run-time иситема — на Си. Компилятор интересен тем, что позволяет подключать функции из системной библиотеки («libc» ☺) статически. Программы компилируются так:

```
obc myprog.mod -o myprog
```

Расширение файла-исходника должно быть «.mod». Ключ «-о» крайне желателен, иначе без него вы получите стандартное имя как в «gcc» — «a.out».

Если необходимо задействовать возможности Оберон-07, то нужно добавить специальный ключ:

```
obc -07 myprog.mod -o myprog
```

Для работы откомпилированных программ необходимо присутствие библиотеки «libffi.so.6» (для версии компилятора 3.1). У меня была установлена библиотека «libffi.so.7» (это файл символьической ссылки), я сделал её копию под именем «libffi.so.6» и мои программы стали без проблем запускаться.

Компилятор предлагается для Linux32, Linux64, Windows32 и Windows64. Есть пакет для RaspberryPI, но для какого именно не совсем понятно. Из исходников его можно собрать и для Мака, но для меня эта тема совсем уж незнакомая и тут я тоже не очень понял, для какого именно.

Component Pascal [54] и **BlackBox** [55]. Попытка скрестить Оберон-2 и Java... 😊 Если кто-нибудь из вас сталкивался с системой «Esmertec JBed», то это ничто иное, как разработка компании Oberon Microsystems. Примерно в 1999 году товарищи разработчики этой системы из Oberon Microsystems ушли, забросив BlacBox, образовав компанию Esmertec. В 2009 году у них произошло слияние с французской Purple Labs, которые занимались той же темой и компания стала называться Myriad Group AG. 98% Java для мобильников — это её продукция. А начиналось то всё с Оберона... 😊

Никто не знает, откуда в названии Component Pascal взялся Паскаль. В интернете некоторое время спорили по этому поводу, но к единому мнению так и не пришли. Сами разработчики молчат об этом как рыба об лёд. Наверное будет не совсем понятным термин «компонентный». Если по простому, то это объединённая концепция динамического ООП и модульных языков, типа Модула или Оберон.

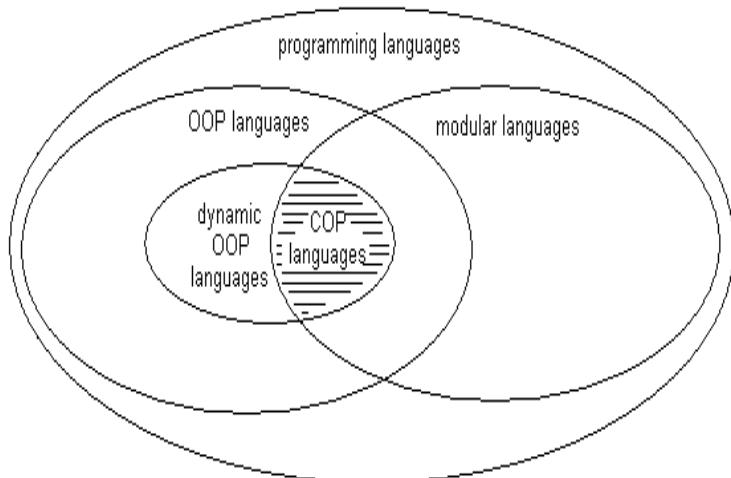


Рисунок 76. Компонентно-ориентированный язык (из книги — Пфистер К. «Компонентное программное обеспечение»)

Здесь термин «динамический» подразумевает подгрузку того или иного компонента только на время его использования, т.е. компилятором он статически с программой не связывается.

После того, как первые разработчики BlackBox ушли из OberonMicrosystems, их последователи довольно долгое время потратили на то, чтобы привязать BlackBox к MicrosoftCOM, продолжая компонентную модель языка. Как выяснилось уже намного позже, время оказалось потеряно зря и лучше бы они занимались привязкой BlackBox к компонентам Java, чтобы на сегодняшний день уже иметь полноценную кроссплатформенную среду программирования.

Самостоятельностью Компонентный Паскаль нигде не замечен. Средой для работы программиста с Компонентным Паскалем служит BlackBox, которая представляет из себя виртуальную машину. Примерно в 2013 году OberonMicrosystems выложила исходники в интернет и с тех пор этой системой занимается мировая программная общественность. Если вы скачаете FreeOberon, то там тоже можно найти BlackBox, однако это будет всего-лишь среда исполнения программ, а не разработки.

Изначально система BlackBox была жёстко привязана к Windows, но после того, как OberonMicrosystems перестала ею заниматься, нашлись энтузиасты, которые попытались сделать версию для Linux, которую можно скачать с [55], ссылка «Скачать». Однако народ, кто с помощью неё пишет в Linux свои программы, пользуется виндовой версией через Wine.

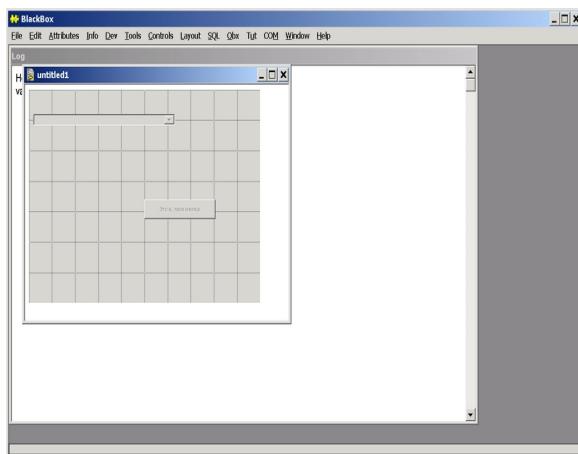


Рисунок 77. В BlackBox можно создавать оконные приложения

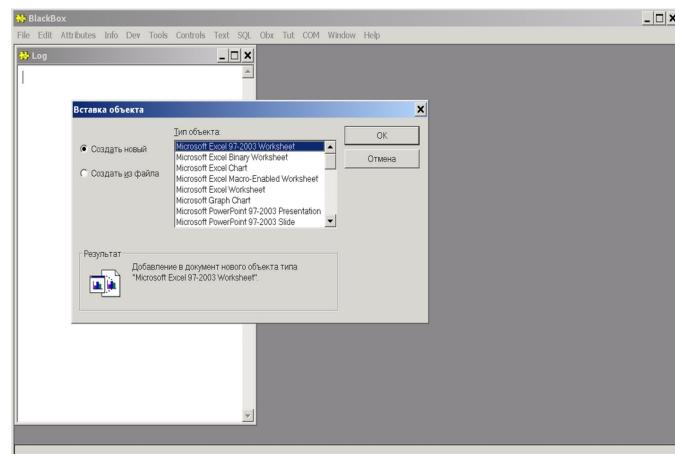


Рисунок 78. А можно вставить какой-нибудь системный компонент

Окошки и все графические элементы в Windows рисуются функциями WinAPI, а в Linux — функциями GTK.

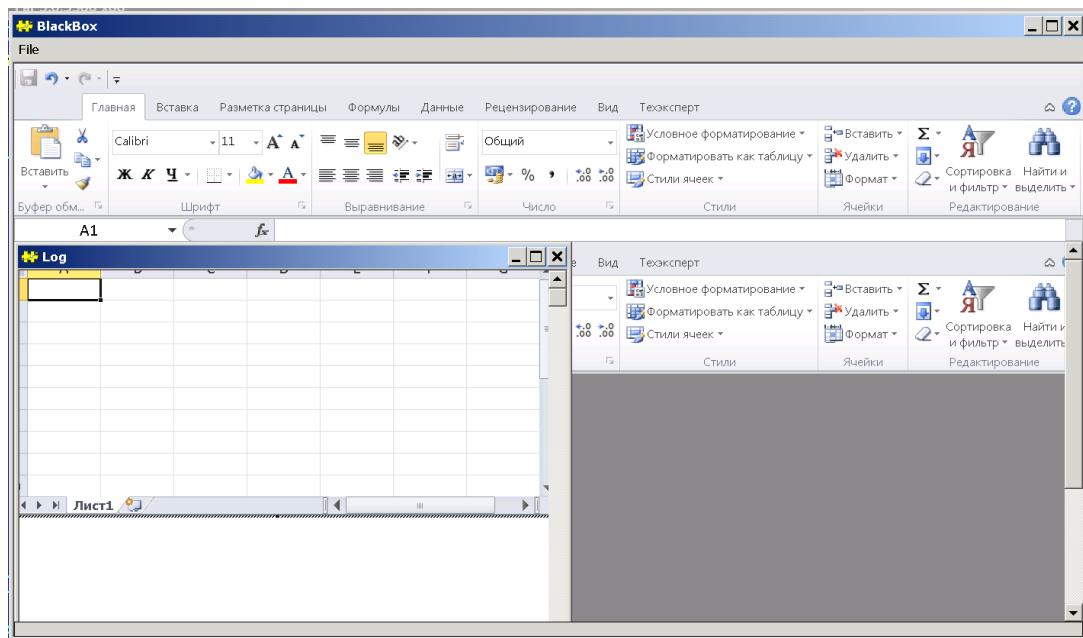


Рисунок 79. Ой, а кто это у нас тут нарисовался... 😊

Как видно из рисунка, самым большим плюсом этой системы можно считать не особо напряжённую возможность писать на оберон-языке оконные приложения. Виднеющееся из-под вновь создаваемой формы окошко «Log», это аналог окна быстрого вывода, которое когда-то было у QuickBasic и FoxPro, т.е. там можно немедленно посмотреть, что делает та или иная конструкция языка.

Несколько напрягает отсутствие русификации в оболочке. Однако это дело легко поправимо — нужно скачать специальный компонент ([56], на страничке есть инструкция по установке), установить и наслаждаться русским языком. BlackBox и Компонентный Паскаль больше известны как среда обучения программированию и, несколько меньше, как среда программирования микроконтроллеров. До меня доходили слухи о какой-то крупной торговой интернет-системе, но более менее внятных следов найти не удалось...

BlackBox может быть не только средой разработки и выполнения кода, но так же делать и самостоятельно выполняющиеся программы. Компиляция кода у него проходит незаметно, однако компоновщик требуется запускать вручную. Будет это выглядеть примерно так:

```
DevLinker.LinkExe dos MyProg.exe := Kernel+ Files HostFiles Console HostConsole MyProg
```

Сложновато, не так ли? 😊 Здесь:

- DevLinker.LinkExe — как нетрудно догадаться, собственно линковщик;
- dos MyProg.exe — опции линковщика, т.е. у нас будет DOS-программа с названием MyProg.exe;
- := — разделитель-указатель, что именно линковщик должен запихивать в MyProg.exe;

- Kernel+ — среда выполнения программы. Естественно не весь BlackBox, а всего лишь Run-Time, т.е. BlackBox урезанный, механизм управления;
- Files HostFiles Console HostConsole — дополнительные модули, откуда берутся процедуры упомянутые в нашей программе;
- MyProg — а это сама наша программа, уже откомпилированная.

Если вы захотите посмотреть исходники программ для BlackBox'a в обычном текстовом редакторе, то увидите в начале и конце кода нечто неописуемое: 😊

```

Zettel0001EbenePara2Norm.Mod — Kate
Файл Правка Вид Проекты Закладки Сеанс Сервис Настройка Справка
Zettel0001EbenePara2Norm.Mod
1 TextDocs.NewDoc<#FCColorFlatLockedControlsOrg:=UBIER>b30#0beron10.Scn.Fnt33MODULE
2 EbenePara2Norm;
3 IMPORT In, Out, Vectors;
4 VAR
5   TaskInfo : Vectors.String;
6   mP : REAL;
7
8 PROCEDURE GetTaskInfo;
9 BEGIN
10   In.Open;
11   In.String (TaskInfo);
12 END GetTaskInfo;
13
14 PROCEDURE CalcNorm;
15 VAR
16   m, vdiff : Vectors.VectorType;
17   i : INTEGER;
18 BEGIN
19   Vectors.VecME := Vectors.VProd (Vectors.VecS1E, Vectors.VecS2E);
20   Vectors.dE := Vectors.Prod (Vectors.VecSE, Vectors.VecNE);
21 END CalcNorm;
22
23 PROCEDURE ShowResults;
24 BEGIN
25   Out.Ln;
26   Out.Ln;
27   Out.String ('Umwandlung Ebenengleichung (Parameter- in Normalenform) fr ');
28   Out.Ln;
29   Out.String (TaskInfo);
30   Out.Ln;
31   Out.Ln;
32   CalcNorm;
33   Vectors.ShowPoint (Vectors.VecSE, 'Sttzvektor (als Punkt) : ');
34   Vectors.ShowPoint (Vectors.VecS1E, '1. Spannvektor (als Punkt) : ');

```



```

Zettel0001EbenePara2Norm.Mod — Kate
Файл Правка Вид Проекты Закладки Сеанс Сервис Настройка Справка
Zettel0001EbenePara2Norm.Mod
33 Vectors.ShowPoint (Vectors.VecSE, 'Sttzvektor (als Punkt) : ');
34 Vectors.ShowPoint (Vectors.VecS1E, '1. Spannvektor (als Punkt) : ');
35 Vectors.ShowPoint (Vectors.VecS2E, '2. Spannvektor (als Punkt) : ');
36 Vectors.ShowPoint (Vectors.VecNE, 'Normalenvektor (als Punkt) : ');
37 Out.String ('Reelle Konstante d der Ebene in der Allgemeinen Normalenform : ');
38 Out.Real (Vectors.dE, 10);
39 OutLn;
40 END ShowResults;
41
42 BEGIN
43   EbenePara2Norm.
44 END EbenePara2Norm.

Kompilieren :
Builder.Compile \s Zettel0001EbenePara2Norm.Mod ~
Systembefreien :
System.Free EbenePara2Norm ~

Aufgaben :
EbanePara2Norm.GetTaskInfo "LS Analytische Geometrie GK; S. 64 Nr. 3a"~
Vectors.GetVecSE 1.0 2.0 0.8~
Vectors.GetVecS1E 1.0 0.0 1.0~
Vectors.GetVecS2E 1.0 2.0 3.0~
EbenePara2Norm.ShowResults ~

Weitere Aufgaben im Zettel0001EbenePara2Norm.Text !
(* 20088222- kpr ... Niklaus' Zettel : Analytische Geometrie - Umwandlung Ebenengleichung
(Parameter- in Normalenform) *)

```

Рисунок 80. Исходники программы в BlackBox

Причём в самом BlackBox'e исходник будет выглядеть нормально. Дело в том, что это не совсем текст, точнее говоря — не один только текст. Туда ещё напиханы параметры окружения из BlackBox. Поэтому, если вы захотите откомпилировать такую программу где-то помимо BlackBox, то текст лучше копировать в текстовый редактор из самого BlackBox'a, либо в текстовом редакторе вырезать всё, что находится до слова «MODULE» и всё, что находится после «END Название_Модуля».

Active Oberon [57]. Это дальнейшая модификация Оберана, сделанная одним из соратников Вирта — Юргом Гуткнхтом. Язык создавался как системный для операционной системы BlueBottle, продолжая виртовскую концепцию языка программирования как видимой верхушки айсберга ОС.

Это ещё одна попытка создать на основе Оберона явно видимый ООП, однако исключительно своими силами, по аналогии с Модула-3, т.е. явным введением в описание языка ссылочного типа OBJECT. Здесь так же разгулялась фантазия автора по типам данных. Типы очень сильно напоминают

фортраньи, правда в отличие от Фортрана здесь появились ещё и беззнаковые числа.

Язык стал несколько более похож на прежний Паскаль. Служебные слова перестали выделяться большими буквами. Если создаётся модуль с объектами для всеобщего пользования, в нём создаются две секции:

- DEFINITION — объявления, которые будут видны другим модулям;
- IMPLEMENTATION — это непосредственная реализация объявленных выше объектов.

Появилась многопоточность и она стала «привязчивой». ☺ Т.е. поток реализован внутри конкретного объекта и не является чем-то отдельным, самим по себе. В Delphi/FreePascal можно обнаружить нечто подобное — есть специализированный класс-поток. Однако там то, что должен делать поток, задаётся в его специальном методе. В Active Oberon объект проектируется со всеми своими действиями, плюс к уже готовому объекту добавляются специальные примочки для компилятора, которые задают алгоритм работы потока. Стоит ещё раз напомнить, что объект в Обероне — это не нечто обёрнутое кодом класса, а модуль целиком. Весь код, который в модуле — это один объект. В него добавляется специальный монитор, для наблюдения за работой потока-объекта, а так же специальные модификаторы, которые указываются внутри фигурных скобок, например {REALTIME}. Такие модификаторы добавляются к разным составным частям объекта, например к переменным или процедурам. Надо сказать, что в процессе разработки и программирования с многопоточностью тут просто так не разберёшься, паскальные или сишиные мерки применять надо с осторожностью.

Из общих характеристик многопоточности: оно стало двух видов — блокирующее вытесняющее и неблокирующее с синхронизацией. Многопоточность в рантайме проще, чем в Паскаль, потому что всю черновую работу берёт на себя компилятор. Но, сами понимаете, сначала нужно тщательное проектирование.

Сборщик мусора выделен в отдельный поток и работает сам по себе, программист ему не требуется. Правда, если вы создаёте потоки реального времени, то такие могут при необходимости останавливать мусорщика, но не навсегда. И злоупотреблять этим не надо... ☺

Zonnon [58]. Разработка того же автора, но только на платформе .NET. В конце 90-ых или начале 2000-ых, Microsoft, в рамках выбора «системного языка» для своей платформы .NET, дала задание ETH провести исследование Модулы\Оберона на предмет возможности «лёгкого» включения в свою ауру. Маркетологам MS тогда что-то не понравилось, видимо посчитали затраты на рекламу нового языка излишними. На основе этой работы в середине 2000-ых был изготовлен сначала проект, а потом и компилятор языка. Что интересно,

компилятор создал наш программист Валерий Зуев.

Zonnon по возможностям является примерной копии предыдущей разработки. Интересен тем, что прекрасно интегрируется в VisualStudio. Впрочем, тут ничего удивительно, так как исходное задание было именно таким. Кроссплатформенность обеспечивается платформой MONO. Компилятор собирается в MS VisualStudio 2012.

Попытка собрать компилятор в MONO окончилась неудачей. Первая проблема оказалась легко решаема — в файлах, где описывались пути к файлам, названия путей\файлов оказались записаны не теми буквами, какими они на самом деле представлены в каталогах. Это исправляется легко — в данном случае я просто переименовал каталоги\файлы такими буквами, как они были представлены в списках.

Вторая проблема оказалась сложнее...

Сборка делалась так:

```
xbuild Zonnon.sln
```

Ошибку можно увидеть на рис. 81. В чём именно заключается проблема — для моего взгляда (одним глазком) так и осталось непонятным... ☺

Код с ошибкой такой:

```
private void VisitInContext(NODE node, XmlNode context)
{
    ...
}

XmlNode await = docSource.CreateNode(XmlNodeType.Element, "await", "");

...

if (node.val != null)
{
    VisitInContext(node.val, await); ← ошибка вот она, закрывающая скобка...
}
```

Рисунок 81. Попытка собрать компилятор Zonnon в MONO...

```

lib.dll
ASTtoXML.cs(1709,46): error CS1525: Unexpected symbol `)'
ASTtoXML.cs(1712,36): error CS1525: Unexpected symbol `)'
ParserN.cs(3176,38): error CS1525: Unexpected symbol `;'
ParserN.cs(3179,49): error CS1525: Unexpected symbol `)'
    Task "Csc" execution -- FAILED
        Done building target "CoreCompile" in project "/home/vadim/projects/c/solution/Compiler/E
TH.Zonnon.csproj".-- FAILED
            Done building project "/home/vadim/projects/c/solution/Compiler/ETH.Zonnon.csproj".-- FAILED
                Task "MSBuild" execution -- FAILED
                    Done building target "Build" in project "/home/vadim/projects/c/solution/Zonnon.sln".-- FAILED
Done building project "/home/vadim/projects/c/solution/Zonnon.sln".-- FAILED

Build FAILED.
Errors:

/home/vadim/projects/c/solution/Zonnon.sln (default targets) ->
(Build target) ->
/home/vadim/projects/c/solution/Compiler/ETH.Zonnon.csproj (default targets) ->
/usr/lib/mono/xbuild/14.0/bin/Microsoft.CSharp.targets (CoreCompile target) ->

    ASTtoXML.cs(1709,46): error CS1525: Unexpected symbol `)'
    ASTtoXML.cs(1712,36): error CS1525: Unexpected symbol `)'
    ParserN.cs(3176,38): error CS1525: Unexpected symbol `;'
    ParserN.cs(3179,49): error CS1525: Unexpected symbol `)'

    0 Warning(s)
    4 Error(s)

Time Elapsed 00:00:01.5777700
Vadim@mycomp1:~/projects/c/solution$
```

Go или **golang**. Приведён здесь исключительно в качестве примера, как можно полностью испоганить идею Оберона. ☺ Попытка скрестить Оберон и Си. ☺ Сначала вам захочется с этим поспорить, но посмотрите на код — и вас не будет покидать ощущение «дежавю», как будто вы это уже где-то видели. ☺ Всё правильно, видели. Половина синтаксиса из Си, а половина — Оберон. Причём основная концепция языка — Обероновская:

- Попытка минимализма. Изначально язык строился на концепции обеспечения работы минимумом языковых конструкций, не дублирующих функционал друг друга;
 - Минимум ключевых слов, простая и легко читаемая грамматическая структура кода;
 - Простота работы с типами данных. Иерархию — в топку;
 - Только явные преобразования;
 - Сборка мусора;
 - Разделение интерфейса и реализации;
 - Система пакетов с явной и чётко указанными зависимостями.

Согласитесь, разве это не Оберон? ☺ Собственно разработчики никогда и не скрывали, что с одной стороны не хотели уходить сильно уж далеко от Си, к которому многие уже привыкли, но были без ума от простоты и ясности Оберона. Таким образом они попытались свести воедино две, казалось бы, несовместимые между собой ветки в программировании:

- Си, который является чуть-чуть более высокоуровневым чем

ассемблер, поэтому позволяет всё, любую шалость. Из-за чего пишущие на нём никогда не вылезают из системных ошибок, даже тщательно следя за ними. Но с другой стороны в его рекламу вложено очень много средств, так что продавить что-то другое сейчас уже и не получится;

- Оберон, который своей простотой и строгостью способен представить программу в её безальтернативном понимании, собственно как и должно быть при реализации идеи.

Как вам этот код? ☺

```
package main

import (
    "os"
    "flag" // парсер параметров командной строки
)

var omitNewLine = flag.Bool("n", false, "не печатать знак новой строки")

const (
    Space = " "
    NewLine = "\n"
)

func main() {
    flag.Parse() // Сканирование списка аргументов и установка флагов
    var s string
    for i := 0; i < flag.NArg(); i++ {
        if i > 0 {
            s += Space
        }
        s += flag.Arg(i)
    }
    if !*omitNewLine {
        s += NewLine
    }
    os.Stdout.WriteString(s)
}
```

Заранее соглашусь с критиками которые скажут, что, судя по внешнему виду кода, Паскаль или Оберон тут и рядом не валялись. ☺ Это так, рядом с таким синтаксисом валялся Си. ☺ Тем не менее, в этом языке 100% лежит именно концепция Оберона, который там возник неспроста. Дело в том, что один из разработчиков языка Go, Роберт Гризмер — выпускник ETH. Как раз он и продвигал Оберон в качестве платформы нового языка. В принципе все с ним были согласны, вот только согласны исключительно со своими фундаментальными добавками. Сам Роберт собирался добавить только пару-тройку фишечек, вроде малюсенького ООП в стиле Вирта, но, увы, победила идея «сделать круче всех, чтобы ого-го!». Как результат — имеем внешний сишный вид, что было бы ещё не сильно большой бедой, но вдобавок ещё и кучу плюшек, сделавших язык уже не таким лёгким и понятным...

В общем и целом про Оберон

Количество трансляторов языка Оберон вовсе не ограничивается теми, что я привёл выше. На самом деле их куда больше. Мои примеры подбирались по принципу минимальной кроссплатформенности, т. е. на основной идее Вирта — простой язык просто реализовать. В сети можно найти ещё с десяток трансляторов исключительно под Windows, что на сегодняшний день уже воспринимается как существенный недостаток.

На форумах мы обычно можем встретить утверждения «Си++ — форева, всё остальное — ацтой!» именно в таком синтаксисе. 😊 Это говорит о том, что все критиканы любых языков критикуют исключительно для того, чтобы громко издать звук, который производят люди, покушавшие чрезмерно много... 😊 О причинах их презрения спрашивать бессмысленно — кровь от мозгов ушла в желудок и далее по организму. 😊

Так жив Оберон или мёртв? Или, если ближе к реальности — делают на нём что-то серьёзное или нет? Как ответ на этот вопрос я специально привёл несколько ссылок ([64] ... [71]), где кратко описывается, в чём именно серьёзном применяют Оберон. Если коротко, то в основном это обучение программированию, наука и микроконтроллеры. Самое серьёзное применение Оберона в России — «Росатом». Но узнать что-то обероноинтересное у этой организации — «оставь надежду всяк сюда входящий...». 😊

На Обероне самое лёгкое обучение программированию. Оберон помогает в науке получать результаты, которые с помощью других коммерчески популярных языков были недостижимы. В [67] профессор Ткачёв привёл интересные примеры, когда с помощью Оберона были не только решены некоторые задачи, которые не решались с помощью других языков, но и парочку коммерчески успешных проектов. Промышленное применение на сегодняшний день — микроконтроллеры приборов, беспилотников, управление на атомных станциях. Поскольку вы не найдёте в сети коды этих решений, поэтому считается, что с Обероном никто не работает.

На мой взгляд, Оберон значительно упростил программирование. Основная фишка Оберона — его понятность для непрограммистов. Не будем забывать, что цель любого языка высокого уровня — быть понятным для непрограммиста. Программисты, если они такие умные, пусть учат ассемблер. Давайте скажем себе честно — учёному или инженеру вовсе не улыбается потратить полжизни на то, чтобы полностью овладеть ещё и каким-то суперязыком программирования для того чтобы решить какое-то квадратное уравнение. Вообще-то им и так есть чем заниматься, поэтому программирование для них должно быть делом необременительным и понятным с первого взгляда. Оберон полностью отвечает этому требованию. Следовательно, если непрограммисту нужно побыстрому посчитать что-то сложное, Оберон — это его всё. На изучение любого другого языка времени

уйдёт оё-ёй....

Что из Оберонов выбрать? В идеале — тот, который соответствует виртовскому Оберон-07(2013) ([60], [61]). В реале же не всё так просто. Большинство трансляторов сделаны на основе стандарта Оберон-2. «Vos» — наиболее кроссплатформенный, сделан на основе стандарта Оберон-2. Универсальный Оберон-07 — безусловно «akron». На сегодняшний день он существует для Windows и Linux. Если нужны окошки — тогда «BlackBox». Правда в Linux его придётся использовать через WINE, поскольку неясны сроки выхода версии Linux-86_x64. К BlackBox'у, на сегодняшний день, написано самое большое количество обероновских модулей и компонентов.

Трансляторы Оберона чаще всего — это транслятор в некий промежуточный код на языке Си или Java, который потом уже компилируется в двоичную программу. Поэтому такой транслятор может работать на всех ОС, где есть GCC или LLVM. Вряд ли преобразование кода в сишный стоит считать недостатком, потому-что по нынешним временам это уже правило — так же работают наборы трансляторов LLVM, GCC и .NET\MONO. Есть трансляторы и в JavaScript, что позволяет использовать Оберон на своих веб-страницах. Единственное что смущает — систему оптимизации компилятора мы используем только сишную. Но и тут нет особой проблемы, потому что система типов Оберона Си-совместима.

Основные претензии к виртовскому Оберону — отсутствие возможности визуального программирования и отсутствие многих тысяч модулей-интерфейсов к сишным библиотекам. Для этого есть очень веские причины. ☺ Во-первых, Обероны — чистые компиляторы. Кто страдает не по детски (☺) по окошкам — можно использовать IDE BlackBox. Окошки там есть, хотя среда программирования сильно проигрывает по количеству разных финтифлюшек таким монстрам как Delphi, Lazarus или MS VS. Чаще всего программирующим на Обероне окошки вообще не нужны, поскольку кроме программирования микроконтроллеров, вторая по распространённости причина использования — всякие разные расчёты. Во-вторых, нет каких либо трудностей подключить ту или иную функцию из сишной библиотеки (GUI или, к примеру, работы с СУБД) к программе на Оберон. Принцип тот же самый, что и в Паскаль. Другое дело, что подключение сишной библиотеки порою не просто не нужно, а ещё и испортит получаемый результат. Прецеденты были, это не голословное заявление.

Ещё одна практически непреодолимая проблема — разработка трансляторов Оберона является чьей-то частной инициативой. Никаких бизнес-интересов за ним не стоит. Первоначальный (и довольно сильный интерес) Microsoft к этому языку разился о невозможность поддерживать ещё один язык в актуальном состоянии, несмотря на его простоту. Вдобавок, на его рекламу потребовалось бы вложить не меньше финанс и усилий, чем до этого в C\C++ или Basic. C# в этом отношении повезло — из Borland удалось в своё время переманить готовую квалифицированную команду. По поводу Оберона

пришлось бы выдержать неслабую конкуренцию с тем же Borland и его хорошо раскрученным продуктом Delphi.

Обероном могла бы заниматься Borland, благо что он довольно близок к Delphi, тем более что у них уже был немалый опыт объединения таких разных языков, как C\С++ и Pascal\Delphi на основе единой визуальной платформы. Однако финансовые дела Borland были намного хуже, чем у Microsoft и заниматься конкурирующим с Delphi проектом они просто не могли. В своё время они уже отказывались от довольно интересного проекта TurboProlog по тем же причинам. Кстати говоря, у Microsoft в своё время была аналогичная ситуация. Когда-то они купили хорошо раскрученный FoxPro и делали на нём неплохие бабки. Однако в эру визуального GUI-программирования посчитали излишним поддерживать два однотипных проекта — Visual FoxPro и Access, прекратив разработку и поддержку первого.

Обероном могла бы заниматься одна из европейских фирм или даже государственное учреждение. В качестве примера можно привести французский проект Scilab, полный аналог американского Matlab, который разрабатывался в INRIA и сейчас используется в качестве инструмента математического моделирования в таких гигантах, как Airbus, Sanofi, Peugeot и ещё в паре десятков не менее заслуженных корпораций. Однако за INRIA стояла и стоит мощная государственная финансовая поддержка и Scilab проектировался и продвигался на государственные деньги. Оберону в этом отношении несколько не повезло. В своё время фирма Oberon Microsystems отпочковалась от ETH в самостоятельную коммерческую организацию. И хотя их разработки finanziровали несколько крупных компаний (в том числе и Borland ☺), однако это финансирование уже не касалось Оберона, а только вполне конкретных коммерческих проектов. С государственной финансовой поддержкой Оберону наверняка повезло бы больше, если бы тот остался под эгидой ETH.

С Active Oberon и Zonnon, несмотря на всю их заявленную разработчиками несокрушимую мощь, в интернете отчего-то дела сложные... Во-первых, довольно регулярно ссылки на них, найденные в интернете, не работают. Если зайти на сайт ETH и попытаться на нём поискать какую-то информацию, ссылки вроде и рабочие, но попасть туда невозможно — почти всегда после очень долгого ожидания загрузки страницы оказывается, что соединение не удалось. Во-вторых, не очень приятные ограничения на среду использования. Active Oberon вы найдёте только для своей собственной ОС A2, других мне отыскать пока не удалось. Я ничего не имею против новой операционки, но хотелось бы и в старых поработать. Хотя изначальной идеей Оберона была его тесная работа с ОС, однако виртовский Оберон сумели сделать отдельным проектом для распространённых операционок, надо бы и с Active Oberon так же сделать. Возможно Юргу некогда, а программирующая общественность подавлена не в меру активными сишиками и пока что чувствует себя неуверенно... ☺

Исходники Zonnon можно взять на Гитхабе, компиляция и использование каких-либо непреодолимых трудностей не вызывает, но здесь мы опять сталкиваемся с ограничением среды работы — только .NET и, в какой-то степени, MONO для Linux. Есть порт MONO для BSD, но поскольку Microsoft этим портом не занимается, то совместимость Zonnon с BSD не ясна. Линуксом фирма Microsoft уже несколько лет занимается довольно ретиво, о чём говорит резкий взлёт функционала MONO и своевременная синхронизация её с платформой .NET, но в MONO отчего-то регулярно перестают или вообще не хотят работать программы, которые прекрасно работают в .NET. Как говорится — думайте сами, решайте сами...



4. Иерархия языков

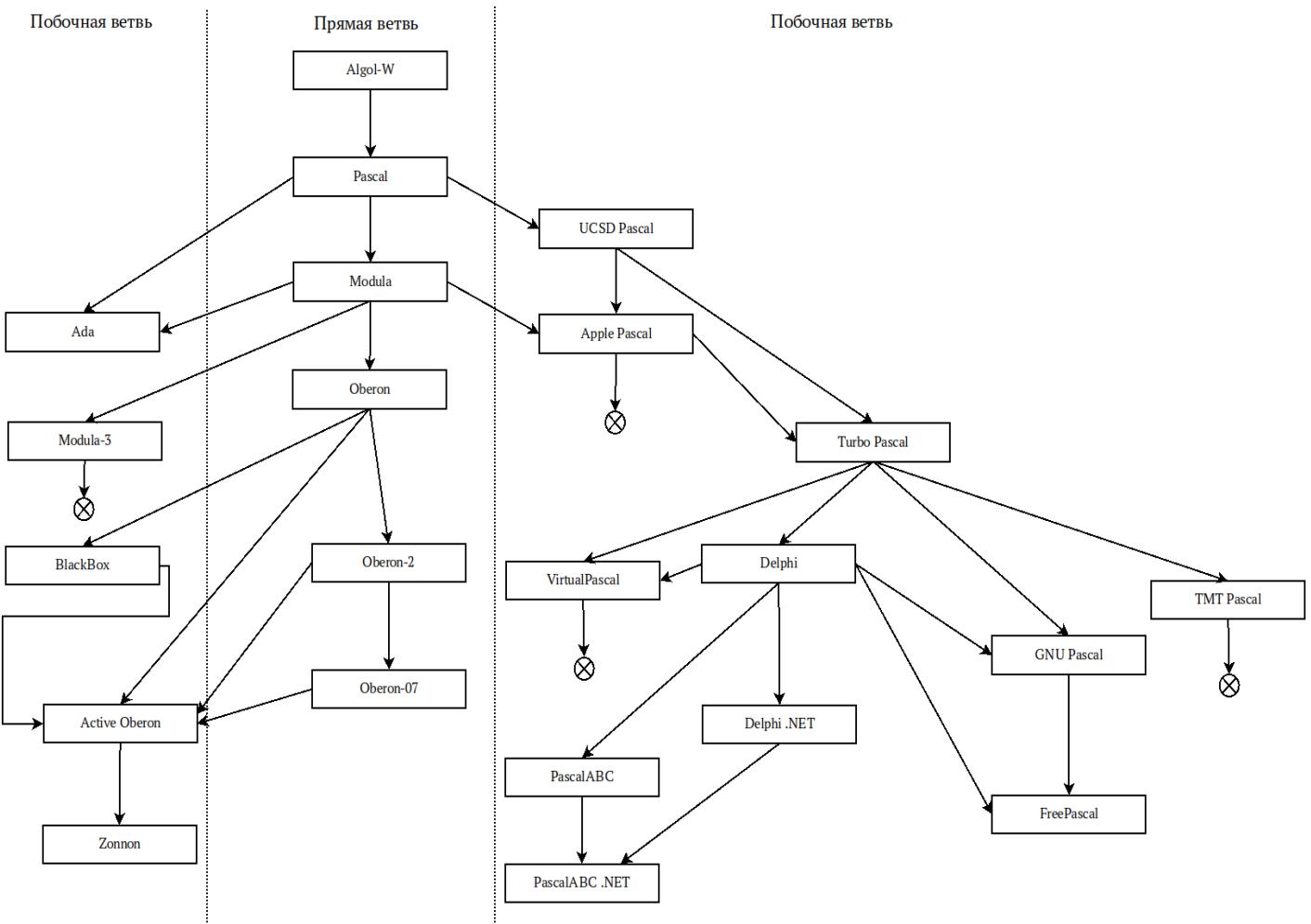


Рисунок 82. Иерархия языков относящихся к Паскалю

Пояснение к рисунку. Прямая ветвь — это всё, что в большей части относится к Вирту. Active Oberon и Zonnon делал уже не Вирт, однако разработчика этих языков Юрга Гуткнехта довольно охотно консультировал. Однако полагая, что такой язык уже приобрёл тяжеловесность, так же как и в случае с Модула-3, личное участие в разработке счёл ненужным.

Там где стрелка идёт из низа одного прямоугольника в верх другого — это прямое наследование. Если же стрелка идёт из боковой стороны и упирается в бок другого прямоугольника — это заимствование отдельных компонентов с той или иной степенью переделки. Там где на рисунке стрелочка упирается в кружочек с крестиком — разработка проекта остановлена.

Я не стал конкретизировать трансляторы Оберона, как это сделано с Паскалем, потому что, в отличие от него, Оберон пока что составляет слишком уж незначительную часть систем разработки программ, даже по сравнению с FreePascal. Поэтому выделить что-то особое довольно трудно.

5. Сравнение производительности

А то как же? Это не мы — жизнь такая... 😊 Без сравнения никак не обойтись. Поскольку у меня в голове ничего кроме численных методов не водится, то не ждите всяких там 3D-тестов с ракетами, молниями и облизывающимися котиками. Для этого лучше обратиться в сторону OpenGL (по-нынешнему — Vulkan) или DirectX. Мы с вами, как это принято в благородном сообществе суперкомпьютеров, порешаем систему линейных уравнений (СЛАУ). Испытываться будут только бесплатно доступные компиляторы, естественно из ныне работающих. 😊

Чтобы сильно не скучать ожидая ответа, возьмём небольшую матрицу 3000 x 3000. Компиляция исходников проводилась без ключей оптимизации. Испытательный стенд — процессор XEON, 8 ГБ ОЗУ. Для компиляторов «voc» и «akron» используются разные названия типов данных матриц (LONGREAL и REAL), но размеры у них одинаковые — 8 байт. Исходники тестов лежат в архиве [77], в подкаталоге «tests». Компиляция исходников производилась без каких либо ключей оптимизации. Результаты представлены в табл. 1

Таблица 1. Результаты тестирования компиляторов на решении СЛАУ

Язык	Компилятор	Время работы, сек.	Примечания
Фортран	gfortran 9.2.1	40	
Си	gcc 9.2.1	91	
Паскаль	FreePascal 3.0.4	111	Использовались динамические массивы
		3.3.1	
	PascalABC.NET 3.6.3	149	
Ада	GNAT 9.2.1	399	1) Linux, консольный компилятор, запуск через MONO; 2) Использовались динамические массивы.
	GNAT 9.2.1	401	Со спецмодулем динамических массивов
Модула-2	gm2 9.2.1	96	Динамический массив создан вручную
	xds 2.51	36	Использовались статические массивы
Оберон-2	voc 2.1	250	Использовались статические массивы
	xds 2.51	35	
	obc 3.1	73	
Оберон-07	akron 1.27	128	

Когда писались программы тестов, меня сначала сильно напрягло, что у Оберонов нет общих с Паскалем средств работы с динамическими массивами. Могло бы показаться, что тесты будут провалены, поскольку все уважающие себя современные трансляторы (не Модула и не Оберон) напрочь отказываются обрабатывать большие статические массивы без применения каких-либо специальных ухищрений. Однако, как оказалось, и Обероны и Модулы без проблем обрабатывают большие статические массивы и всякие там STACK_OVERFLOW не возникают.

FreePascal по скорости работы уже где-то возле GCC гуляет. Если у кого-то из разработчиков хватит мужества заняться оптимизацией генерируемого двоичного кода, то скоро можно будет на все претензии сишиков говорить: «Наездникам черепахи — физкульт привет!». ☺

Меньшая скорость работы PascalABC.NET наверное связана с работой через дополнительную прослойку MONO. Но и здесь скорость неплоха.

Очень удивил результат «xds», который обогнал даже Фортран. Похоже в нём разработчики хорошо поработали над оптимизацией компилятора. Если бы кто-нибудь взялся за продолжение разработки и перевёл бы обероновскую часть на стандарт Оберон-07, это был бы самый лучший в мире транслятор общего назначения.

Порадовал результат моего любимого «akron». На форуме разработчик заявлял, что над оптимизацией он совсем не работал. Однако, как видно и без оптимизации получается хороший результат по скорострельности. Как видно упрощение Оберон-07 по сравнению с Оберон-2 пошло исключительно на пользу.

Результат «GNAT» тоже меня удивил, но только с другой стороны. По правде говоря, я ожидал от него скоростей лежащих рядом с «gm2», потому что они работают на одной платформе. Но поскольку я не специалист по Аде, возможно где-то в коде допустил какую-то принципиальную ошибку.

6. Ссылки и книги

1. [Niklaus Wirth, 1984 ACM Turing Award Recipient](#)
2. [Энциклопедия языков программирования](#)
3. [Википедия. Язык ассемблера](#)
4. [В.В. Броль, А.А. Красилов, А.Н. Маслов - Язык программирования АЛГОЛ 68, Итоги науки и техн. Сер. Теор. вероятн. Мат. стат. Теор. кибернет., 1978, том 15, 163–232](#)
5. [Association for Computing Machinery \(ACM\)](#)
6. [ACM Transactions on Mathematical Software \(TOMS\)](#)
7. Йенсен К., Вирт Н. - Паскаль. Руководство для пользователя и описание языка/Пер. с англ., предисл. и послесл. Д. Б. Подшивалова - М: Финансы и статистика, 1982. - 151 с, ил.
8. Тумасонис В., Дагене В., Григас Г., Паскаль. Руководство для программиста: Справочник: Пер. с литовск. - М: Радио и связь, 1992. - 192 с.: ил.
9. [A. M. Addyman, R. Brewer and more - A Draft Description of Pascal](#)
10. Языки программирования Ада, Си, Паскаль. Сравнение и оценка/Под ред. А. Р. Фьюэра, Н. Джехани: Пер. с англ. под ред. В. В. Леонаса. М: Радио и связь, 1989. - 383 с.: ил.
11. [ISO 7185:1983 Programming languages — PASCAL](#)
12. [ISO 7185:1990 Information technology — Programming languages — Pascal](#)
13. [Pascal. ISO 7185:1990 \(текст документа\)](#)
14. [OberonCore. Библиотека](#)
15. [Beat Heeb, Immo Noack - Hardware description of the workstation Ceres-3](#)
16. [Lola-2: A Logic Description Language](#)
17. [Pascal for small machines](#)
18. [DEPARTMENT OF DEFENSE. REQUIREMENTS FOR HIGH ORDER COMPUTER PROGRAMMING LANGUAGES "STEELMAN". June 1978](#)
19. [ГОСТ 27831-88. Язык программирования АДА](#)
20. [www.adacore.com](#)
21. [Download GNAT Community Edition](#)
22. [The GAP community. Over 200 members in 35 countries teaching Ada and SPARK using GNAT.](#)
23. [Язык программирования Ада](#)
24. [Ada Programming](#)
25. [Википедия. Turbo Pascal](#)
26. [Lazarus снапшоты](#)

27. [MSEide+MSEgui](#)
28. [The fpGUI Toolkit Project](#)
29. [PascalABC.NET. Современное программирование на языке Pascal](#)
30. [MODULA-2 Home page](#)
31. [Free Modula-2 Pages](#)
32. [ISO/IEC 10514-1:1996 Information technology — Programming languages — Part 1: Modula-2, Base Language](#)
33. [А. Ф. Пар - СТАНДАРТ ЯЗЫКА МОДУЛА-2. Пособие для изучения. Электронная версия с исправлениями и уточнениями](#)
34. [The home of the GNU Modula-2 compiler](#)
35. [Исходники gm2 \(GNU Modula-2\)](#)
36. [Oberon Revival. XDS-компилятор Модула-2/Оберон-2](#)
37. [XDS Модула-2/Оберон-2](#)
38. [XDS Modula-2/Oberon-2 Development System for Windows](#)
39. [Excelsior IDE for XDS Modula-2 Developers](#)
40. [Resource Page @modula3.org \(formerly m3.org\)](#)
41. [Никлаус Вирт - Modula-2 и объектно-ориентированное программирование](#)
42. [theForger's Win32 API Tutorial - Version 2.0](#)
43. [MODULA-2. Chapter 15 - Example programs](#)
44. [Персональная страничка Никлауса Вирта на ETH про язык Оберон](#)
45. [Oberon Mycosystems](#)
46. [Вики BlackBox Component Builder](#)
47. [www.projectoberon.com](#)
48. [Х. Мёссенбёк, Н. Вирт - Язык программирования Оберон-2](#)
49. [Free Oberon. Сайт проекта](#)
50. [Вадим Исаев - Сборка Lazarus из исходников](#)
51. [Vishap Oberon Compiler](#)
52. [Oberon-07 compiler for x64 \(Windows, Linux\), x86 \(Windows, Linux, KolibriOS\), MSP430x{1,2}xx, STM32 Cortex-M3](#)
53. [Hanspeter Mössenböck - Object-Oriented Programming in Oberon-2, Second Edition](#)
54. [Сообщение о языке Компонентный Паскаль](#)
55. [BlackBox Framework Center](#)
56. [BlackBox. Русское меню и документация](#)
57. [Сообщение о языке Active Oberon](#)
58. [Р. О. Митин - Язык программирования Zonnon \(основы\)](#)
59. [Е. В. Касьянова - Язык программирования Zonnon для платформы Microsoft.NET](#)
60. [Niklaus Wirth - The Programming Language Oberon \(Revision 1.10.2013 / 3.5.201\) \(Oberon-07\)](#)
61. [Никлаус Вирт - Язык программирования Оберон \(Ревизия 1.10.2013 / 3.5.2016\) \(Оберон-07\)](#)

62. wiki.oberon.org
63. [ИНФОРМАТИКА-21. МЕЖДУНАРОДНЫЙ ОБЩЕСТВЕННЫЙ НАУЧНО-ОБРАЗОВАТЕЛЬНЫЙ ПРОЕКТ](#)
64. [Дмитрий Викторович Дагаев - Перспективы Оберон-технологий для стратегических областей](#)
65. [Иван Денисов - Оберон-технологии на службе биофизики](#)
66. [Ермаков И. Е. - Инструменты семейства Оберон \(языки, среды, фреймворки\)](#)
67. [Ермаков И. Е. - Системный анализ качеств Оберон-технологий в контексте классов задач цифровой индустрии](#)
68. [Фёдор Васильевич Ткачёв - Оберон — «серебрянная пуля»](#)
69. [Фёдор Васильевич Ткачёв - Оберон в физике элементарных частиц](#)
70. [Совместная лекция Фёдора Ткачёва и Юрга Гуткнхта "Проект Оберон: 30 лет"](#)
71. [Иван Денисов - Большое количество видеолекций и уроков для начинающих по Оберон-07 и BlackBox](#)
72. [Ada он-лайн](#)
73. [FreePascal он-лайн](#)
74. [Оберон он-лайн](#)
75. [Карло Пешио - Никлаус Вирт о культуре разработки ПО](#)
76. [Oxford Oberon-2 compiler](#)
77. [Примеры программ](#)

Приложение. Список программ входящих в архив с примерами

convertor

conv1.pas Конвертор синтаксических единиц исходников программ языка Pascal в язык Oberon.

examples

oberonbyexample Исходный код примеров по всем ключевым аспектам Оберон-2 от разработчиков Vishap Oberon. Взято с ихнего GitHub'a.

Areas.mod Пример для Модула-2 (проверялось на компиляторе gm2). Вычисление площадей разных фигур;

call_uname.adb Пример использования системной функции из системной биб-ки для языка Ада;

call_uname.ob07 Тоже самое для Оберон-07 (akron);

dmi_vendor.adb Пример чтения системных файлов для Ада;

Math Дополнительные математические функции для компилятора akron (Оберон-07), которые отсутствуют в стандартной поставке.

MathBits.ob07 Битовые операции с целыми числами;

MathExt.ob07 Знак числа, квадрат и квадратный корень;

MathGeom.ob07 Некоторые функции связанные с геометрическими расчётами;

MathRound.ob07 Дополнительные функции округления, аналоги того, что есть в FreePascal\Delphi;

MathStat.ob07 Элементарные статистические функции;

Rand.ob07 Генератор случайных чисел, алгоритм Леиьера. Нужен для предыдущего модуля.

tests Тестовые программы измерения скорости вычислений для языков, упомянутых в книжке. Решение СЛАУ методом Гаусса.

gauss1.adb Для компилятора gnat (язык Ада), версия 1;

gauss2.adb Для компилятора gnat (язык Ада), версия 2;

gauss.c Для компилятора gcc (язык Си);

gauss.f90	Для компилятора gfortran (язык Фортран);
gauss_gm2.mod	Для компилятора gm2 (язык Модула-2);
gauss.mod	Для компилятора obc (язык Оберон-2);
gauss.ob	Для компилятора voc (язык Оберон-2);
gauss.ob07	Для компилятора akron (язык Оберон-07);
gauss.ob2	Для компилятора xds (язык Оберон-2);
gauss.pas	Для компилятора fpc (язык Паскаль);
gauss_xds.mod	Для компилятора xds (язык Модула-2);
RandomArr.ob07	Для компилятора akron (язык Оберон-07, дополнительный модуль с генератором случайных чисел).