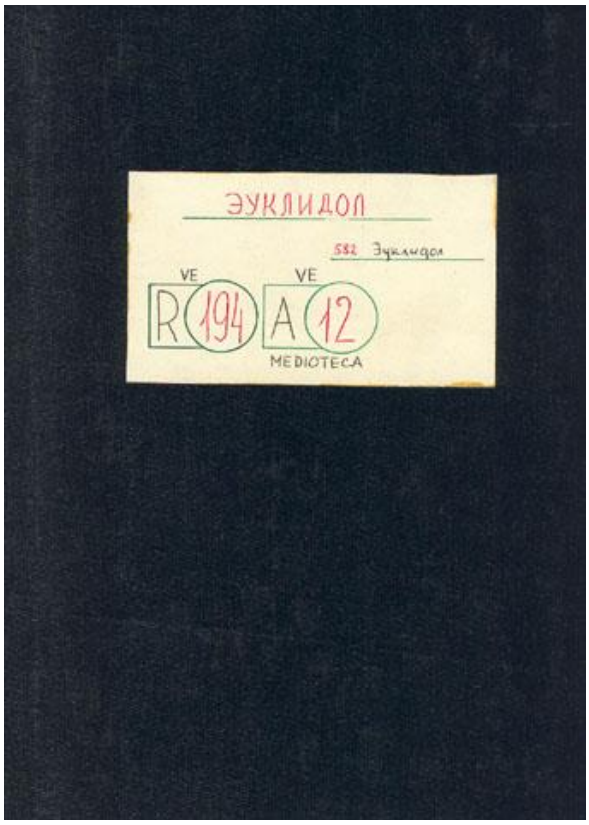
Дневник

Quod sentimus loquamur,
quod loquimur sentiamus!

VEcordia

Извлечение R-NATUR2



Обложка сборника «Эуклидол»
в Третьей Медиотеке

Открыто: 2007.01.14 00:48
Закрыто: 2008.10.24 22:40
Версия: 2018.07.27 23:20

ISBN 9984-9395-5-3
Дневник «VECORDIA»

ISBN 9984-688-34-8
Валдис Эгле. «О природе чисел»

Валдис Эгле

ЭУКЛИДОЛ

Сборник «О природе чисел»
Часть 2-я

Impositum

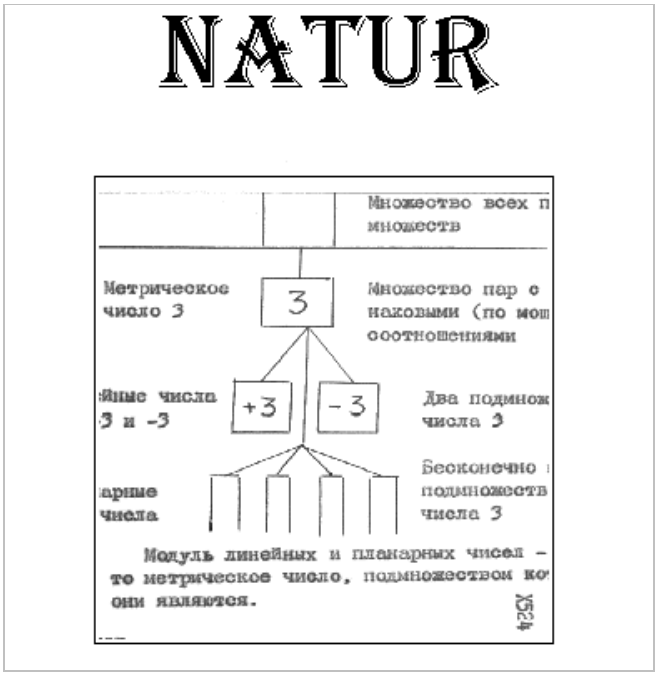
Grīziņkalns 2018

Talis hominis fuit oratio,
qualis vita

(Первая часть сборника «О природе чисел» в {R-NATUR1})

Предисловие сборника «Эуклидол»

1980.05
(раньше на 8 месяцев)



Логотип книги NATUR («О природе чисел») в Шестой Медиотеке

.987. Настоящий сборник содержит три медитации из цикла «Механика Идей»:

ЭУКЛИДОЛ;
ПРЕДИКАТ;
АЛГОРИТМ.

.988. Медитации этого сборника объединяет то, что они являются первыми работами по языку Эуклидолу. Язык описывается параллельно с его транслятором и интерпретатором – системой Эуклидос.

.989. В медитации ЭУКЛИДОС излагаются общие принципы организации системы программ «Эуклидос».

.990. В медитации ПРЕДИКАТ рассматривается синтаксис языка Эуклидола и аппарат «Предикат» системы Эуклидос, который производит синтаксический анализ текста на Эуклидоле.

.991. В медитации АЛГОРИТМ описан первый рабочий аппарат системы Эуклидос (и языка Эуклидола), при помощи которого можно описать и реализовать линейные алгоритмы работы с множествами (дедуктивные алгоритмы отражения).

.992. Аппарат «Алгоритм» не является ни единственным, ни главным из аппаратов Эуклидоса и Эуклидола. В дальнейшем будут рассмотрены и другие аппараты.

.993. Сущность системы Эуклидос состоит в том, что она моделирует на ЭВМ некоторые механизмы человеческого мозга, задействованные в процессе отражения и мышления. Разумеется, нет однозначного соответствия между тем, что происходит в Эуклидосе, и тем, что происходит в голове человека. Но на данном этапе развития науки, которая пока еще не в состоянии точно сказать, что происходит в голове человека, когда он мыслит, попытка догадаться об этом на примере ЭВМ – не последнее средство.

7. Тетрадь EUKLS

ЭУКЛИДОС
Моделирующая мышление система программ

Lutum, nisi tunditur, non fit testa
(Не замесив глины, не вылепишь горшка)

Написано: 1980.01 – 1980.04 Рига

Медия EUKLS (в Третьей Медиотеке медитация ЭУКЛИДОС) содержит общее описание проекта компьютерной системы, моделирующей человеческое логическое мышление.

1. Цель медитации ЭУКЛИДОС

1980.02
(раньше на 3 месяца)

.994. Настоящую работу можно считать прямым продолжением медитации ТЕОРИКА. Там я поставил перед собой вопрос о том, как нужно излагать теории.

.995. Главными достижениями, сделанными до сих пор в этом направлении, я считаю:

.996. а) аксиоматический метод, впервые широко примененный Эвклидом;

.997. б) овладение после Кантора столь глубоким и фундаментальным понятием, как множество;

.998. в) полную формализацию языка, предпринятую Гильбертом в логике предикатов.

.999. Свои размышления я считаю одновременно и продолжением этих дел (в направлении, которое мне показалось правильным) и критикой их недостатков.

.1000. Мне хотелось иметь язык, универсальный для всех наук от экономики до математики, полностью формализованный, говорящий о множествах и позволяющий отличить действительные основания от позднейших выводов теории. Этот гипотетический язык я назвал Эуклидолом в честь первого из трех основателей упомянутых направлений.

.1001. С настоящей медитации я начинаю построение этого языка.

.1002. Главным недостатком всех трех упомянутых достижений я счел игнорирование действительной сущности мышления человека. Действительная же сущность мышления по моему мнению заключается в том, что это мозг человека обрабатывает информацию о внешнем мире. Эту информацию он может обрабатывать только по каким-то алгоритмам. Эти алгоритмы и есть абстрактные теории. Следовательно, язык описания теорий должен быть языком описания алгоритмов или, иными словами, алгоритмическим языком.

.1003. Значит, согласно этим представлениям, Эвклид, Кантор, Гильберт и их многочисленные последователи, сами того не ведая, пытались создать средства описания алгоритмов. Мне эти средства показались мало приспособленными к такой цели. Мне хотелось побольше использовать здесь современную технику, теорию и опыт алгоритмических языков, созданных для компьютеров.

.1004. Поскольку, согласно этим представлениям, в мыслях человека нет вообще ничего другого, кроме информации о внешнем мире, алгоритмов ее обработки и результатов этой обработки, то люди, естественно, ничего другого и не могут друг другу передавать, когда они каким-нибудь способом общаются между собой. Следовательно, всякий язык (в том числе русский) – это алгоритмический язык.

.1005. Человеку, который это слышит впервые и раньше об этом глубоко не задумывался, такие утверждения, что всякая теория – набор алгоритмов и всякий язык – алгоритмический язык, пожалуй, покажутся глупой дикостью. Меня, профессионального программиста, могут

обвинить в том, что я абсолютизирую свою науку компьютеров, механически распространяя ее понятия на все остальные науки. Сам же я думаю, что моя профессия во всем этом сыграла только ту роль, что мне, профессиональному программисту, было несколько легче, чем, скажем, профессиональному математику или профессиональному лингвисту, увидеть не только внешнее, но и глубинное сходство между теорией и программой, между русским и Алголом, а роль ЭВМ заключалась в том, что они показали простые варианты того, что в сложной форме происходит в мозге, и заставили людей придумать такие простые средства описания алгоритмов, что я со своим Эуклидолом мог идти уже по протоптанной дороге.

.1006. Итак, **краеугольным камнем построения языка изложения теорий был вывод теорики о том, что теория – это набор алгоритмов**, а язык ее описания – алгоритмический язык. Справедливость этого тезиса я больше не буду защищать никакими философскими или логическими аргументами. Не может быть лучшего доказательства этого, чем теория, изложенная на алгоритмическом языке, поэтому довольно философии и вперед к Эуклидолу!

.1007. Цель настоящей медитации – изложить некоторые самые начальные и самые общие соображения об этом языке, которые послужат основой к более детальному его изложению в дальнейших медитациях.

2. Как описывать алгоритмические языки

1980.02

.1008. Поскольку Эуклидол является языком алгоритмическим, то не будет неуместным здесь немного задуматься о том, что же это такое – алгоритмический язык – и как его описывать.

.1009. Под алгоритмическим языком я понимаю систему соглашений по кодировке и дешифровке на промежуточном носителе алгоритмов, а также их материалов и продуктов, с целью передачи информации о них от одного субъекта к другому. Если принимающим субъектом является ЭВМ, то язык называется языком программирования. Таким образом, языки программирования – это подмножество алгоритмических языков.

.1010. Кодирование информации проводится при помощи **знаков**. Не следует думать, что знак – это только что-то написанное на бумаге; знаком может быть и пробивка в перфокарте, и частота звуковой волны, в общем: всё, что угодно, лишь бы это было материальным. Множество знаков называется **текстом**. Сущность общения посредством языка состоит в том, что текст, закодированный одним субъектом, декодируется другим.

.1011. Значит всякий алгоритмический язык (в частности, и Эуклидол), его место и связи нужно рассматривать по следующей схеме (см. Фиг.1 {.1024}): имеется предмет (теория) в материальном мире, от которого к субъекту А поступает информация, которую тот обрабатывает согласно своим алгоритмам (теории) и выдает какой-то результат. Если субъект А закодирует свои алгоритмы (теорию) в текстах языка (например, Эуклидола), если субъект В декодирует эти тексты и если при этом их соглашения по языку будут совершенно полными и четкими, то субъект В сможет воссоздать у себя алгоритмы (теорию) настолько точно, что всегда при поступлении от предмета одинаковой информации к субъектам А и В они будут получать одинаковые результаты. Точность и полноту языка, как и корректность работы субъекта А при кодировке и субъекта В при дешифровке текста, можно проверить идентичностью результатов.

.1012. Из этой схемы следует, что рассмотрение сущности всякого алгоритмического языка (и Эуклидола в частности) будет более менее полным только тогда, если будут раскрыты по крайней мере пять общих вопросов:

.1013. 1) Каким путем и какая информация поступает от предмета к субъектам А и В.

.1014. 2) Как осуществляется обработка этой информации по заданному алгоритму внутри субъектов А и В и какие получаются при этом результаты.

.1015. 3) Каким образом субъект А кодирует алгоритмы.

.1016. 4) Каким образом субъект В производит дешифровку.

.1017. 5) Какие связи и закономерности действуют в текстах (теория текстов).

.1018. Когда говорят «описание или изучение алгоритмического языка», обычно на самом деле имеют в виду лишь последний пункт: теорию текстов на данном языке, при этом полагая, что часть материала, относящегося к первым четырем пунктам, известна, а часть не нужна.

.1019. Когда я был студентом университета, нам начали преподавать алгоритмические языки еще до знакомства с самими ЭВМ, то есть, пользуясь приведенной здесь терминологией,

стали преподавать теорию текстов при абсолютном отсутствии знаний по остальным четырем пунктам. Этот университетский курс алгоритмических языков я помню как бредовый кошмар. На экзамене я получил свою первую тройку в университете. Может быть, моя неприязнь к языкам программирования высокого уровня была бы меньше, если бы не эта ошибка моих университетских преподавателей.

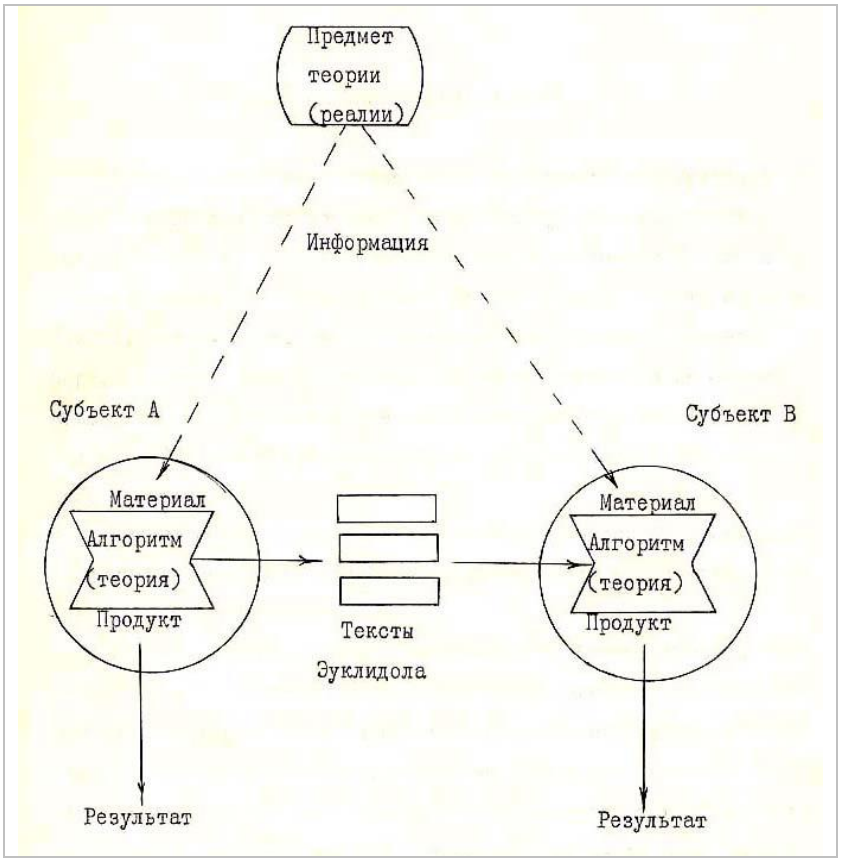
.1020. Какая-то абстрактная бессмыслица, абсолютно лишенная привязанности к чему-либо материальному, что-то такое, что невозможно понять, а можно лишь тупо зазубрить – такими предстали передо мной алгоритмические языки, поданные как чистая теория текстов без всякого понимания того, что и зачем, собственно, делается. Только значительно позже, когда я узнал, как функционируют ЭВМ, как они получают и обрабатывают информацию, как выдают результаты, как они дешифруют то, что я написал на своем алгоритмическом языке, – только тогда я понял, что такое алгоритмический язык.

.1021. Не знаю, как у других людей, но лично для меня все абстрактные и обобщенные описания алгоритмических языков программирования, языков управления заданиями, входных потоков различных программ (редакторов, библиотекарей и т.д.) и сегодня продолжают оставаться туманными и расплывчатыми до тех пор, пока я не узнал (или не догадался), как работают соответствующие программы, что и над какими объектами они делают, то есть, пока я не познал тех алгоритмов и механизмов, которые скрываются за общими словами. Поэтому лучшими описаниями я всегда считал те, которые прямо рассказывали об этих механизмах, о действиях этих трансляторов, редакторов и библиотекарей.

.1022. Да и вообще, я здесь, как и везде, продолжаю оставаться неисправимым материалистом, держусь за материю, чтобы не «уплыть в туманную даль», и стараюсь избежать абстрактных, отвлеченных слов, хочу внести во всё это как можно больше материального, телесного, осязаемого, реального, физического в виде тех механизмов, которые работают в субъектах A и B.

.1023. Итак, стремясь не повторить то, что нахожу ошибкой своих университетских преподавателей, и стараясь внести в обсуждение языка как можно больше конкретного, я не мог ограничиться изложением одних только правил Эуклидола (теории текстов), как обычно делаются при описании алгоритмических языков; я считал абсолютно необходимым включить в описание также и хотя бы часть других перечисленных выше пунктов.

.1024.



Фиг.1. Общение на алгоритмическом языке

3. Машинная реализация языка

1980.02

.1025. Теперь передо мной стала новая проблема: язык Эуклидол предназначен для письменного общения между людьми, значит, как субъектом А, кодирующим текст, так и субъектом В, его читающим, должен быть человек. Но что я могу сказать о механизмах, которыми человек обрабатывает информацию о внешнем мире, кодирует свои и дешифрует чужие алгоритмы? Я не сомневаюсь, что эти механизмы реальны и материальны, но, увы, я, как и вообще вся современная наука, могу сказать о них очень мало конкретного.

.1026. Значит полная программа из пяти пунктов описания Эуклидола пока не выполнима? Придется ограничиваться одной только отвлеченной теорией текстов?

.1027. И тогда я решил сделать так, чтобы субъектом В могла быть и ЭВМ, то есть, сделать Эуклидол не только алгоритмическим языком общения людей, но и неким языком программирования. В отличие от механизмов человека, механизмы ЭВМ я могу описать не с бесконечной, но всё же с очень большой подробностью. Разумеется, механизмы в человеке и в ЭВМ отличаются, но между ними должно быть и что-то общее, раз они имеют одинаковые результаты.

.1028. Это была не единственная причина, по которой я решил непременно сделать машинную реализацию субъекта, читающего тексты Эуклидола.

.1029. Всем, работающим с ЭВМ, хорошо известно, что в текстах, анализируемых ЭВМ, не может быть абсолютно ничего неоговоренного, неформализованного, значит машина – надежный судья полной формализации языка. Подобный язык, все соглашения которого полностью оговорены, и который поэтому понятен даже ЭВМ, я называю фундаментально точным.

.1030. В-третьих, машинная реализация играла для меня примерно такую же роль, какую играла арифметическая реализация аксиоматических систем для Гильберта: разве может быть лучшее доказательство непротиворечивости и реальности всего, о чем я рассуждаю, чем воплощение этих идей в железе и переливающихся потоках электрических импульсов? Одно дело, когда я говорю: «существует такой-то алгоритм и абстрактное множество, заданное им», но совсем другое дело, когда я могу небрежно сказать: «у меня есть программа, которая всё это запросто проделывает на ЭВМ».

.1031. Итак, язык Эуклидол предназначен для людей, но ввиду трех главных причин в его построении очень важное место занимает машинная его реализация:

.1032. а) для получения возможности изучить язык более глубоко, чем в виде простых закономерностей в текстах;

.1033. б) для контроля строгости формализации;

.1034. в) для доказательства непротиворечивости и реальности рассуждений.

.1035. Как видите, среди этих целей нет таких, как доказательство при помощи ЭВМ каких-то новых истин, построение новых теорий и т.д.

.1036. Можно придумать много различных алгоритмических языков изложения теории. Эуклидол – лишь один из возможных и, так как для меня он был первым, то имеется мало оснований надеяться, что он окажется хорошим. Эти сомнения, однако, не касаются самой идеи описания теорий алгоритмическим языком.

.1037. Аналогично после того, как избран язык, можно еще сделать много различных его реализаций на ЭВМ. Начиная с этой медитации, я опишу одну из таких возможных систем программ – ту, которую я в действительности составил для достижения упомянутых выше целей. Я назвал ее системой Эуклидос (EUKLIDOS).

.1038. Поскольку цели этой системы, которые я только что назвал, были столь ограниченными, то в Эуклидосе я не старался обеспечить максимально много удобств пользователю, обеспечить практически неограниченную длину внутреннего файла хранением его на диске, создание на диске библиотеки алгоритмов и другие средства, которые сделали бы Эуклидос похожим на солидные современные программные системы. Наоборот, я старался минимальными затратами труда и времени получить проверку Эуклидола на ЭВМ, пусть ценой очень существенного ограничения возможностей системы.

.1039. Аналогично при выборе машины и операционной системы единственным критерием для меня было то, на какой машине и под какой операционной системой мне это легче и быстрее сделать.

.1040. Всегда, когда передо мной становилась дилемма «программировать хорошо» или «программировать быстро», я выбирал путь «быстро», применял более простые методы и отказывался от пусть изящных, но сложных в реализации алгоритмов.

.1041. Всегда, когда передо мной становилась дилемма «программировать так, чтобы программа работала быстрее и эффективнее или так, чтобы она требовала меньше памяти», я выбирал вариант «меньше памяти».

.1042. Это будет полезно знать тому, кто захочет оценить мои программистские решения с профессиональной точки зрения.

.1043. Итак, во всей этой разработке главная цель – это язык, предназначенный для письменного общения между людьми при изложении ими теорий. Эуклидос – лишь вспомогательное средство.

.1044. На первый взгляд кажется, что в этой ситуации я должен был сначала и в основном рассматривать язык Эуклидол, а описание системы Эуклидос привести лишь как приложение где-то в конце.

.1045. Описывая язык, я вынужден определять различные термины, связанные с этим языком, такие, например, как «символ», «высказывание» и т.д. Но из тех идей, которые я в общих чертах описал уже в ТЕОРИКЕ и которые в дальнейшем разверну еще подробнее, вытекает, что определение термина, это на самом деле не что иное, как описание алгоритма, по которому данный объект можно выделить из множества всех других доступных объектов. Если же я подробно опишу алгоритм, по которому Вы можете из текста на Эуклидоле выделить отдельные символы, высказывания и т.д., то это окажется практически описанием соответствующей подпрограммы Эуклидоса. Таким образом, уже при разговоре о «чистой» теории текстов я не могу отделить описание Эуклидола от описания Эуклидоса.

.1046. Тем более трудным становится отделение этих двух описаний, если я (согласно принятой выше методологии) хочу рассмотреть язык не только как закономерности в текстах, но и в связи с теми действиями, которые совершаются в субъектах, работающих с языком.

.1047. Поэтому я решил описывать Эуклидол и Эуклидос параллельно во времени и рядом в пространстве (в соседних главах). Общее правило таково: сначала в одной главе дается описание какой-то части языка (как теории текстов), а в следующей главе – описание соответствующего аппарата Эуклидоса.

4. Как описывать Эуклидос

1980.02

.1048. Такая постановка выдвинула две новые проблемы. Первая заключается в вопросе о том, с какой точностью и подробностью описывать систему Эуклидос. В программистском мире существуют такие стандарты, как описание для системного программиста, описание для пользователя, общее описание и т.д.

.1049. Многие на моем месте, пожалуй, в такой работе, как эта, предназначенной в основном для изложения языка, а не собственно для описания системы программ, ограничились бы какими-нибудь общими описаниями. Но я избрал другой путь: в конечном виде описания даются с максимальной подробностью и точностью, предназначенной для прямого разбора листингов (распечаток) программ, которые также входят в полное описание. Я сам наиболее высоко всегда ценил именно такие описания, и теперь оборачиваю этот критерий к себе, считая, что описания всегда могут оказаться недостаточными, но никогда не могут быть слишком подробными и точными.

.1050. Эуклидол – язык точности, так пусть уж его собственные описания и описания его машинной реализации будут достойны его.

.1051. Но здесь появляется обычная проблема, преследующая всех программистов: полностью, исчерпывающе и адекватно (так, чтобы абсолютно всё в описании соответствовало действительности) можно описать только программу, разработка которой окончательно

завершена. Если же система программ продолжает развиваться, то обычно имеет место одно из двух:

.1052. а) либо при том, что первые версии системы уже вовсю эксплуатируются, отсутствует сколь-нибудь подробные их описания;

.1053. б) либо тщательно и прекрасно оформленное описание уже устарело, не успев дойти до читателя, так как в самой системе уже что-нибудь изменилось, и читателю никогда точно не известно, что, собственно, в описании устарело.

.1054. Итак: либо мириться с отсутствием подробных описаний, либо тратить много труда на заведомо устаревшие описания.

.1055. Ни то, ни другое меня не устраивало. Я хотел сохранить себе право и возможность непрерывно пополнять и расширять Эуклидос, и при этом всё время иметь полные, исчерпывающие и совершенно точно соответствующие действительности описания. Обычно эта проблема частично решается при помощи аппарата версий. Но на самом деле никаких четких границ версий не существуют, изменения обычно вносятся очень небольшими порциями и непрерывно.

.1056. Решить эту проблему можно только в одном случае: если описания обладают такой же модульностью и гибкостью изменения, какой обладают сами программы. А для этого нужно, чтобы описания создавались теми же средствами, что и программы, а именно: машиной. (*Во время создания этой медитации в Латвии, во всяком случае в окружении автора, еще не было ни одного компьютерного документатора – ред.*).

.1057. Поэтому я в Эуклидосе встроил Автодескриптор – программу, описывающую Эуклидос. В машинописных медитациях дается лишь общее описание системы, которое должно рассматриваться как введение к описаниям Автодескриптора (полагается, что после прочтения медитаций по Эуклидосу Вы перейдете к чтению описаний Автодескриптора и, наоборот, что, читая автоскрипты Эуклидоса, Вы уже знакомы с общими положениями, изложенными в медитациях).

.1058. Тот, кто желает работать с Эуклидосом, в первую очередь должен попросить систему описать себя. Таким образом Вы всегда будете иметь гарантию, что у Вас в руках описание именно того Эуклидоса, с которым Вы имеете дело.

.1059. Никакие версии Эуклидоса не отслеживаются и не нумеруются, всё ориентировано на то, что у Вас описание именно Вашего Эуклидоса, состоящее из текстов медитаций плюс тексты автоскриптов.

.1060. Вторая проблема, возникающая при описании Эуклидоса, заключается в том, что читатель, желающий побеседовать об основаниях теорий, логике предикатов, теории множеств и подобных абстрактных вещах, вдруг оказывается погруженным в мир компьютеров, в мир, который ему может быть совершенно чужим и непонятным.

.1061. Избрав такой способ описания языка (через Эуклидос) я поставил читателя перед необходимостью читать подробное описание программы, написанной для ЭВМ IBM/360 на Ассемблере/360, если он хочет познакомиться с языком Эуклидолом, который в общем-то не имеет никакого отношения к IBM/360 и Ассемблеру. Такое мое поведение могут признать следствием дурного тона или непродуманности, так как подобные описания обычно считаются доступными только программистам, да и то только узкой специальности. У меня же нет никакой гарантии, что читатель вообще знаком с программированием, тем более на Ассемблере IBM/360.

.1062. Я приношу Вам, мой читатель, свои глубокие и искренние извинения. Конечно, кое-что должно остаться Вам непонятным, если Вы в глаза не видели ЭВМ (кое-что останется неясным и тому, кто сам программировал на Ассемблере/360), но я не вижу способа лучше, чем этот. Если бы я излагал то же самое «общечеловеческими» словами, не прибегая к терминам науки компьютеров и Ассемблера, Вам всё равно многое осталось бы неясным и неточным (я думаю, что даже больше, чем сейчас). Зато теперь хоть кто-нибудь сможет понять это с достаточной глубиной. Да и Вы, если пропустите «белиберду» про всякие там фиктивные секции и базисные регистры (что сделать не очень трудно), и если будете смотреть на всё это как на описание действий, которые должны провести Вы, именно Вы, над потоком текста Эуклидола, то по-моему вполне сможете понять главное.

.1063. Даже если я избавил бы Вас от знакомства с ЭВМ при описании синтаксиса языка, то всё равно невозможно уклониться от этого в дальнейшем. Основная идея, начинающаяся в ТЕОРИКЕ и всё сильнее развертывающаяся в дальнейших медитациях, состоит в том, что теория – это алгоритмы. Предметом нашего разговора являются алгоритмы. Но этот разговор был бы

очень расплывчатым и неопределенным, если бы мы не знали точно кем, как и над каким материалом алгоритм может быть реализован.

.1064. Одна глубина продуманности у умозрительного рассуждения, и совсем другая – при написании и отладке программ.

.1065. Поэтому я не могу отказаться от машинной реализации всего того, о чем рассуждаю; это судья и доказательство строгости и реальности моих построений. «Держись за материю, чтобы не уплыть в туманную даль!» – таков всегда был мой девиз {ROAD.112}, и вот я держусь за материальную машину. Поэтому так или иначе, но тому, кто пожелал познакомиться с моими рассуждениями, придется окунуться в среду ЭВМ.

.1066. Что поделаешь – мы живем в веке компьютеров и скоро знание основ программирования будет считаться таким же элементарным требованием к образованному человеку любой профессии, как теперь, скажем, владение арифметикой. (*В то время в школах еще не преподавали информатику – ред.*).

.1067. К сожалению или к счастью, но невозможно обо всем этом говорить, не влезая в ЭВМ. Но как же быть с теми читателями, кто с компьютерами все-таки не знаком?

.1068. На решение или смягчение этой проблемы направлены три предпринятые в этих медитациях мероприятия:

.1069. 1) Концепция алгоритмического языка излагается в таком общем виде, который никак не связан не только с какой-нибудь конкретной ЭВМ (что обычно), но и вообще никак не связан с ЭВМ (что вряд ли обычно), а примеры приводятся из таких областей, которые знакомы всем. Это мероприятие, кроме облегчения чтения для неспециалистов по ЭВМ, преследует еще и цель показать всеобщность алгоритмического подхода и вездесущность алгоритмических языков.

.1070. 2) В медитацию вставлена глава с кратким описанием машины IBM/360, на которой функционирует Эуклидос. Если читатель пожелает познакомиться с этим поближе, то я могу ему порекомендовать книгу[1] системного аналиста Кларенса Б. Джермейна «Программирование на IBM/360», которая вышла в 1967 году в Нью-Джерси на английском языке и в 1971 году (первое издание) в русском переводе в издательстве «Мир» в Москве.

.1071. 3) Описания языка и программ, хоть они и расположены рядом, я стараюсь организовать в отдельные главы так, что у читателя, категорически не желающего читать подробные описания программ, пожалуй, не может быть трудностей в том, чтобы эти места пропустить, раз уж его удовлетворяет отсутствие у него подробных сведений о работе той или иной части Эуклидоса. Названия глав помогут ему в этом, прямо или косвенно указывая, относится ли глава к языку Эуклидолу или к системе Эуклидос.

5. Трансляторы и интерпретаторы

1980.02

.1072. Теперь посмотрим подробнее, как происходит дешифровка текста алгоритмического языка в субъекте B (то есть, в читателе).

.1073. Представьте себе, что я выписал из школьного учебника по алгебре правила (алгоритмы) составления и решения самых простых алгебраических уравнений и задачку о яблоках разной стоимости или о поездах, движущихся друг другу навстречу, записал всё это на каком-то языке на бумагу и подал этот листочек в окошко специального заведения, например ИРПЗ (Институт Решения Простых Задачек). Через некоторое время мне через это же окошко вернули листочек с решением, записанным на каком-то языке.

.1074. Итак, произошло общение между двумя субъектами (мною и ИРПЗ) на каком-то алгоритмическом языке. Я могу рассматривать весь институт ИРПЗ как некий преобразователь листочков (точнее – текстов на языке), который преобразовал мой текст в свой ответ. Я могу рассматривать ИРПЗ и как некий интерпретатор, который применил поданные ему правила составления и решения уравнений к также поданной ему конкретной задачке о яблоках. Тогда ту часть текста, где записаны эти правила, я буду называть **программой** для интерпретатора, а ту часть, где записана задачка о яблоках – **данными**.

[1] Джермейн Кларенс Б. «Программирование на IBM/360». Мир, Москва, 1971.

.1075. Может быть, приемщица, сидящая непосредственно за окошком Института, сначала перевела мой текст на внутриинститутский язык, прежде чем отправить по пневматической почте на верхние этажи ИРПЗ. Я могу тогда рассматривать ее как некий транслятор, преобразующий текст с языка, принятого в интерфейсе окошка в язык, принятый в интерфейсе пневматической почты, и считать, что весь интерпретатор ИРПЗ распадается на транслятор первого этажа и на собственно интерпретатор, находящийся на верхних этажах.

.1076. Но нет, конечно, гарантии, что ИРПЗ, как все заведения решения простых задачек, не страдает бюрократией. Может быть, на втором этаже мой текст снова переводят на какой-нибудь язык, прежде чем послать на третий этаж. Таким образом получается целый **каскад** трансляторов и интерпретаторов, которые вложены друг в друга.

.1077. Однако каскад сопряженных интерфейсами трансляторов где-то должен кончаться, иначе я бы никогда не получил ответа. В конце концов мой преобразованный текст должен попасть к «настоящему» интерпретатору, который и решает задачку. Посмотрим, чем же он отличается от транслятора.

.1078. Транслятор работает так: берет из ящика стола свою «Инструкцию по переводу текстов», берет мой текст и применяет эту инструкцию к моему тексту. Значит транслятор – это такой же интерпретатор, только у него программа своя, а весь мой текст – данные. У того же интерпретатора, который я счел «настоящим», и программа, и данные – из моего текста. Как видите, разница между транслятором и интерпретатором относительна (лишь по отношению к определенному тексту). Точно так же относительна разница между программой и данными в моем тексте, так как, с одной стороны, всё это – данные для транслятора, а, с другой стороны, мои данные можно рассматривать как программу для некоторого интерпретатора, в который входит весь ИРПЗ плюс моя программа.

.1079. Поскольку транслятор – такой же интерпретатор, то и он может иметь иерархическую структуру и каскад трансляторов, переводящих Инструкцию.

.1080. Теперь посмотрим, как работает интерпретатор, осуществляющий либо мою, либо свою программу. Программа состоит из **операторов**, то есть, закодированных приказов что-то сделать. Например, программа транслятора, переводящего мой текст на язык второго этажа, может содержать такие операторы: «взять с полки словарь», «найти в нем нужное слово», «записать перевод на листочек» и т.д. Для того, чтобы выполнить каждый оператор, интерпретатор делает целый ряд более элементарных **операций**, к примеру, для выполнения оператора «взять с полки словарь» нужно встать со стула, подойти к полке, вытащить словарь, вернуться к стулу и сесть на него. Но очевидно, что в каждую из этих операций опять входит целый ряд еще более элементарных операций, например, чтобы встать со стула, нужно послать импульс спинным мышцам расслабиться, мышцам на животе сократиться (после чего тело нагибается вперед), потом привести в действие мышцы ног и т.д. Но что такое «послать приказ мышцам»? Это значит выставить напряжение на нейроны номер... и снять напряжение с нейронов номер... Конец этой цепочки всё более элементарных операций ускользает куда-то за горизонты квантовой механики.

.1081. Итак, выполнение каждого оператора программы – это опять иерархия операций, спускающаяся ко всё более элементарным операциям, пока мы (весьма произвольно) не объявим какие-то операции абсолютно элементарными и неделимыми.

.1082. После того, как в «Инструкции по переводу текстов» подробно описано, как нужно переводить одно слово, можно не повторять это относительно других слов, можно вернуться по инструкции назад, сказав: «то же самое сделать с остальными словами». Таким образом этот участок программы интерпретатор выполнит многократно (в цикле). Сами же действия интерпретатора разлагаются в последовательный ряд (во времени).

.1083. Не исключено, что трансляцию на первом и на втором этаже ИРПЗ осуществляет физически один и тот же интерпретатор, только следуя разным инструкциям (под управлением разных программ). Следуя традиции, я буду в этом случае говорить, что это работают две разные программы вместо того, чтобы говорить более точно, что работает один интерпретатор под управлением двух разных программ. Еще более неточным (но удобным) является утверждение, что эти две программы и есть те самые два транслятора. На самом деле транслятор (то есть, тот, кто осуществляет трансляцию, перевод) – это интерпретатор вместе с программой, а не одна только программа.

.1084. Не исключено также, что в ИРПЗ вообще все преобразования текстов между интерфейсами осуществляет один единственный интерпретатор, и каскады трансляторов на

самом деле предназначены именно для того, чтобы перевести мой текст на язык этого интерпретатора.

.1085. Итак, рассматривая общение двух субъектов (например, меня и ИРПЗ) при помощи текстов на каком-нибудь языке, я буду:

.1086. 1) Рассматривать главный интерфейс (окошко) с данным субъектом или главным интерпретатором (ИРПЗ).

.1087. 2) Внутри субъекта (главного интерпретатора) рассматривать ряд внутренних интерпретаторов, связанных между собой внутренними интерфейсами.

.1088. 3) Считать, что информация через интерфейсы передается в виде текстов (множеств знаков) и что в каждом интерфейсе существуют свои соглашения о значении этих знаков, то есть – свой язык.

.1089. 4) Различать в каждом интерфейсе тексты программ и тексты данных, и, следовательно – соглашения о кодировке программ и кодировке данных (язык программ и язык данных).

.1090. 5) Считать, что программа состоит из ряда операторов, каждый из которых может выполняться интерпретатором многократно.

.1091. 6) Считать, что интерпретатор выполняет операторы последовательно, что выполнение каждого оператора представляет собой последовательность элементарных операций, то есть, что всё выполнение программы – это тоже линейная последовательность действий, но что разные интерпретаторы могут работать параллельно.

.1092. 7) Рассматривать в каждом интерфейсе следующие вопросы:

.1093. а) язык данных (каким образом в интерфейсе представлены данные);

.1094. б) язык программы (каким образом в интерфейсе представлены программы);

.1095. в) управление данными (откуда интерпретатор знает, какие данные брать);

.1096. г) управление операциями (откуда интерпретатор знает, что делать дальше).

.1097. Читатель, конечно же, не подумал, что всё здесь сказанное и все приведенные здесь термины относятся только к Институту Решения Простых Задачек.

.1098. Именно по этой схеме я буду рассматривать реальный и конкретный субъект – Эуклидос: как иерархию вложенных интерпретаторов, с внешними и внутренними программами, состоящими из операторов, реализация которых требует иерархию всё более элементарных операций.

.1099. Как видите, всё это на самом деле абсолютно не предполагает наличия ЭВМ и относится вовсе не к науке компьютеров, а к науке обработки информации вообще. ЭВМ – лишь частный случай общих принципов.

.1100. Тем не менее, я считаю своим долгом поговорить и об ЭВМ.

6. IBM/360

1980.01
(раньше на 1 месяц)

.1101. Оперативная память ЭВМ представляет собой море ферритовых сердечников, каждый из которых путем посылки через него определенного электрического тока может быть намагничен в одном из двух противоположных направлений. Программисты обычно забывают про сердечники и токи, и когда они намагничивают (точнее: намагничивает машина под управлением их программы) этот сердечник в одном направлении, то они говорят, что записали в БИТ (так они именуют сердечник) единицу, а когда намагничивают его в противоположном направлении, то говорят, что записали туда ноль.

.1102. Бит слишком маленький, чтобы с ним было удобно работать как с самостоятельной единицей памяти, поэтому биты объединяются в одинаковые группы (на разных машинах по-разному; в IBM/360 по 8 битов) и каждой такой группе битов присваивается свой номер (адрес). Зная этот адрес, можно однозначно идентифицировать группу битов, читать оттуда или записывать туда информацию. Группа из восьми битов называется байтом, а группа из 32 битов – словом.

.1103. В байт можно записать 2^8=256 различных комбинаций единичных и нулевых битов, в слово – соответственно 2^{32}. Что означает каждая комбинация битов, зависит от их интерпретации машиной, то есть, от договоренности между машиной и человеком. Например, байт 11010010 машина в зависимости от ситуации может интерпретировать как букву «К», как число 210 в двоичной позиционной системе счисления, как приказ ей скопировать один участок памяти в другой участок («переслать», как говорят программисты) или же каждый бит может означать что-то отдельное. Мнение о том, что машина оперирует числами {ROAD.929}, является распространенным, но печальным заблуждением. На самом деле машина, как и человек, оперирует электрическими импульсами и их следами в своей памяти.

.1104. Оперативная память бывает различной емкости. Ее измеряют в единицах К. Единица К равна 2^{10}=1024 байта. Реальны такие длины памяти, как 64К, 128К, 256К, 512К и 1024К, то есть, в большинстве случаев оперативная память состоит из сотней тысяч байтов. Если средняя страница книги содержит 2000 букв и других знаков, включая пробелы (40 строк по 50 букв), то, записывая один знак в байт, машина с памятью 512К может запомнить небольшую книжку размером около 250 страниц, если кто-нибудь ее набьет на карты и введет в машину.

.1105. Кроме оперативной памяти, в которой хранятся программы, исходные данные и результаты, с которыми машина в данный момент работает, она имеет процессор, который, собственно, и выполняет команды, читая их одну за другой из оперативной памяти, и имеет различные внешние устройства, которые либо записывают информацию в память, либо читают ее оттуда. Главными внешними устройствами являются:

.1106. а) магнитные диски, куда информация любого типа (программы, данные, результаты) из оперативной памяти записывается на длительное хранение, чтобы потом ее восстановить в оперативной памяти;

.1107. б) ввод перфокарт, записывающий информацию, набитую на картах (80 знаков на одной карте);

.1108. в) печатающее устройство (принтер), которое, получив определенную комбинацию битов, печатает на бумаге тот или иной знак (128–132 знака в строчке);

.1109. г) дисплей, который отображает знаки на экране и может ввести то, что люди набирают на его клавиатуре (*когда писалось данное сочинение, в Латвию только начинали поступать первые дисплеи – ред.; – сам автор к дисплею доступа не имел; работал только с перфокартами, что и отражается в тексте, например, в пункте* {.1104});

.1110. д) пишущая машинка, которая и печатает, и вводит с клавиатуры информацию.

.1111. На магнитных дисках информация доступна только самой машине. Это ее долгосрочная память. Как в магнитофоне можно сменить пленку, так на машине можно снять диск и поставить другой. Обычно, если машина не предназначена специально для хранения и обработки огромных массивов информации, на ней могут быть установлены одновременно диски с общей емкостью от нескольких десятков до нескольких сотен миллионов байтов (около сотни книг по 500 страниц). Ну, а количество дисков вообще, как и пленок у любителя магнитофонных записей, не ограничено и вполне реальна общая емкость в несколько миллиардов байтов.

.1112. На клавиатуре дисплея или пишущей машинки непосредственно в зале ЭВМ, когда работает программа, много информации не прочитаешь и не наберешь; эти устройства предназначены для обмена между человеком и ЭВМ оперативными сообщениями и директивами. Основную массу новой для ЭВМ информации человек заранее подготавливает на картах и потом вводит в машину с недоступной для человека скоростью. Аналогично при выводе основную массу выводимой информации машина с недоступной для человека скоростью печатает на принтере, а человек потом, уйдя с машины, спокойно разбирает эту распечатку.

.1113. Но можно организовать работу на машине и по-другому: например, к машине присоединены очень много дисплеев, за которыми люди работают с удобной для них скоростью, а ЭВМ обслуживает всех их одновременно.

.1114. То, какую машина имеет память и какие устройства к ней присоединены, называется конфигурацией машины.

.1115. С общей точки зрения все внешние устройства машины – это преобразователи информации или трансляторы, связанные с машиной определенным интерфейсом.

.1116. Сама машина на самом деле – чрезвычайно универсальный интерпретатор, способный выполнять очень разнообразные программы, состоящие из операторов (команд) более, чем сотни разновидностей. Эти программы первоначально обычно подготавливают люди – программисты.

.1117. Программисты обладают всеми недостатками обычных людей, в частности, их язык (или жаргон) чрезвычайно неточен. Они говорят как об одной и той же программе о весьма разнообразных объектах: о текстах, существующих в совершенно разных интерфейсах на совершенно разных языках. Так, например, они утверждают, что текст, который они написали на бланках, текст на перфокартах, тексты во многочисленных интерфейсах трансляторов в оперативной памяти и на дисках, текст на листинге и текст, который во время выполнения читает из оперативной памяти машинный интерпретатор – всё это одна и та же программа.

.1118. Всем здравомыслящим людям совершенно ясно, что работать может только машина, тем не менее программисты, вопреки очевидной логике, утверждают, что работает и выдает результаты их программа.

.1119. Как настоящий программист, я буду допускать все эти неточности.

.1120. Программа ЭВМ во время ее выполнения представляет собой набор битов. Машина последовательно просматривает этот набор. Прочитав первый байт, она интерпретирует его как команду (например, так: «ага, команда номер 1А, значит, сложить два регистра!»). Потом она читает следующий байт и интерпретирует его как два четырехбитовые номера регистров (регистры – это тоже наборы битов; всего в машине 16 регистров с номерами от 0 до 15). Тогда она берет указанные два регистра, интерпретирует их биты уже как числа, складывает, и результат записывает обратно в один из регистров. После этого машина выбирает очередной байт и опять интерпретирует его как новую команду.

.1121. В принципе так же машина работает и с данными, находящимися не в регистрах, а в оперативной памяти. Только в отличие от адреса (номера) регистра, адрес (номер) байта памяти она получает из команды не сразу, а в два приема: сначала номер регистра (он называется базисным), где хранится начальный адрес – (базис) какого-нибудь поля, потом смещение нужного байта в этом поле (поле – это набор байтов). Какими регистрами базировать какие поля, управляет сам программист. Один базис позволяет достигать участок памяти длиной 4К. Такая организация, во-первых, экономит память на хранение адресов, а, во-вторых и главное, позволяет одной программе работать уже с другим полем, изменив базис.

.1122. Вся работа ЭВМ состоит из таких вот действий, предписанных программой, то есть, алгоритмом, который закодирован в наборе битов по определенным соглашениям. Такими же наборами битов являются и данные – исходная информация программ, и результаты ее работы, которые уже интерпретирует человек.

.1123. В общем безразлично, в каком месте памяти будет находиться программа, и, если программист не очень страдает любовью к порядку, то он может в значительной степени перемешивать куски программ с рабочими полями и с исходными данными – машине это безразлично, если только в конце каждого куска программы стоит команда перейти к другому куску.

.1124. Конечно же, для машины нет никаких проблем выполнить один и тот же кусок программы много раз (обычно каждый раз с другими исходными данными). В хорошей программе такие куски, выполняющие четко определенную работу, обычно оформляются как подпрограммы. Это означает, что в тот момент, когда какая-нибудь другая программа осуществляет переход к этой подпрограмме (вызывает ее), в определенном месте (в памяти или регистрах) будет находиться информация, которую подпрограмма должна расшифровать по четко оговоренным соглашениям, а результат своей работы разместить опять по таким же соглашениям (эти соглашения называются соглашениями интерфейса подпрограммы), а потом передать управление обратно той программе, которая ее вызвала. Работу такой подпрограммы можно рассматривать и анализировать независимо от того, кем, когда и зачем она будет вызвана, и ее «внешний мир» ограничивать только соглашениями интерфейса. Именно так я и буду часто поступать.

.1125. Новую программу программист пишет на бумаге ручкой на специальном языке, который отличается от человеческого только простотой. Потом он сам или специальные девушки набивают этот текст на картах или набирают на клавиатуре дисплея и вводят в машину. Там эта программа, называемая исходной, представлена в виде набора битов, где каждый байт кодирует одну букву или другой знак.

.1126. Этот набор битов поставляется в качестве данных различным ранее созданным разработчиками машины программам (трансляторам), которые в конце концов создают другой набор битов, уже пригодный для того, чтобы машина непосредственно интерпретировала их как

команды. Такая уже готовая программа копируется (записывается) на магнитный диск и хранится там подобно тому, как песня хранится на магнитофонной ленте.

.1127. Когда программист (или кто-нибудь другой) хочет выполнить эту программу, он дает приказ специальным программам скопировать данную программу с диска в оперативную память (загрузить) и передать ей управление (то есть, начать выполнять ее), подав ей исходные данные точно так же, как ее исходный текст когда-то был подан программе-транслятору.

.1128. Большие программы (системы программ) обычно состоят из отдельных более или менее самостоятельных частей, называемых модулями. Общее определение модуля (как и программы) никто не в состоянии дать; обычно всё зависит от программиста – он сам решает: «вот здесь еще у меня один модуль, а здесь уже другой». По многим причинам выгодно программы писать так, чтобы модули как можно меньше зависели друг от друга.

.1129. Большая программа может работать значительное время, и нет необходимости все ее части всё время держать в оперативной памяти. Можно постоянно иметь в памяти только те части программы, которые работают часто, а остальные копировать с диска только тогда, когда появляется в этом необходимость (всё это делают автоматически сами программы), причем загружать в одно и то же место. Это позволяет экономить оперативную память. Части, которые находятся в оперативной памяти всё время, пока работает данная программа, называются резидентными, а части, которые загружаются по необходимости – транзитными.

.1130. Почти все программы, написанные не разработчиками машины, работают под управлением той или иной операционной системы (это большой комплекс «старых», давно написанных, программ). Это значит, что новая программа уже на этапе ее подготовки транслируется, редактируется и помещается на диск программами этой операционной системы, а в дальнейшем опять программами этой же операционной системы загружаются в оперативную память и в процессе выполнения она обязательно обращается к операционной системе за услугами.

.1131. Двумя главными операционными системами на IBM/360 являются ОС (большая) и ДОС (небольшая операционная система). Программы, предназначенные для одной системы, не могут работать под управлением другой. Можно написать и такие программы, которые после того, как они записаны на диск, больше не нуждаются в помощи операционной системы. Такие программы называются системно-независимыми.

.1132. В исходной программе программист присваивает различным битам, байтам и полям определенные имена, при помощи которых потом указывает транслятору, что действия надо производить именно с этим битом или байтом. Эти названия предназначены для транслятора, однако удобно ими же обозначать эти биты и байты также и в описаниях программы. Более того, удобно даже саму информацию, которая была или будет или может быть записана сюда, обозначать этим же именем. Так это название превращается в обозначение, код абстрактного параметра, которым характеризуется сам объект, информация о котором может быть записана в память. Этим приемом я часто буду пользоваться.

.1133. Так что, если читатель увидит дальше слова, например, «байт ХС», то пусть он не думает, что речь идет о каком-то таинственном байте ХС, о котором может знать только избранник, постигший все глубины устройства машины. Речь идет о самом обыкновенном байте памяти, которому я присвоил имя ХС, чтобы впредь не путать его со всеми остальными байтами.

.1134. Если читатель понял всё, что написано здесь, то, по-моему, он на самом деле уже обладает всеми знаниями, необходимыми для того, чтобы понимать дальнейшее, за исключением некоторых действительно второстепенных деталей, относящихся к специфике устройства машины и языка Ассемблера, на котором написаны исходные тексты Эуклидоса. Эти детали он с чистой совестью может опустить.

7. Физическая структура Эуклидоса

1980.02
(через 1 месяц)

.1135. Язык Эуклидол, как я уже говорил, предназначен для письменного общения между людьми и, так сказать, канонической формой Эуклидола является текст, написанный на бумаге. Именно таков главный интерфейс субъектов, общающихся на Эуклидоле.

.1136. Если из этого главного интерфейса текст направляется к Эуклидосу, то сразу можно выделить иерархию внутренних интерпретаторов в субъекте (см. Фиг.2 {.1159}):

.1137. 1) Главный интерпретатор (№1) – вычислительный центр. Здесь текст Эуклидола сразу преобразовывается транслятором №1 (перфораторщица с перфоратором) на язык следующего интерфейса (№2 – перфокарты). Это преобразование простое: каждому письменному знаку ставится в соответствие комбинация пробивок в одной колонке перфокарты.

.1138. 2) Второй интерпретатор – машина. Новым транслятором (№2) (устройство ввода карт) создается текст третьего интерфейса – текст в оперативной памяти. Преобразование опять простое: каждой комбинации пробивок на перфокарте ставится в соответствие определенная комбинация битов в байте.

.1139. 3) Третий интерпретатор – система программ. Можно считать, что система программ состоит из операционной системы и собственно Эуклидоса, и рассматривать операционную систему как третий транслятор. Преобразование состоит в отсеивании некоторых карт, не доходящих до Эуклидоса, и в переносе (возможно) карт с одного места памяти на другое.

.1140. 4) Четвертый интерпретатор – собственно Эуклидос. Каскад трансляторов на этом не кончается, но дальнейшую иерархию я рассмотрю позже {.1489}.

.1141. Если присмотреться к первым трем трансляторам попристальней, то можно обнаружить, что они сами распадаются на каскады трансляторов, между которыми существуют еще новые интерфейсы. Но я удовлетворюсь приведенной здесь структурой, так как первые трансляторы для этой медитации не представляют практически никакого интереса.

.1142. Трансляторы №1 и №2 – это два физически разных интерпретатора, в то время, как транслятор №3 и интерпретатор №4 в конечном счете реализуются одним и тем же интерпретатором (процессором машины), работающим с разными программами.

.1143. Таковы первые ступени иерархии интерпретаторов в субъекте В при машинной реализации Эуклидола. Теперь рассмотрим Эуклидос как систему (множество) программ.

.1144. Эуклидос функционирует на машинах системы IBM/360 и на ЭВМ, программно совместимых с ними. Он написан на Ассемблере/360.

.1145. В первом приближении Эуклидос состоит из:

а) Супервизора (модуль А);
б) Резидента (модуль В);
в) рядовых или рабочих модулей (С, Е, ... и т.д.).

.1146. Схему см. Фиг.3 {.1160}. Из этой схемы видны основные принципы логики физической организации Эуклидоса:

.1147. 1) Вся работа с внешней средой сконцентрирована в Супервизоре; только он олицетворяет весь внешний мир для всех остальных модулей. Все рядовые модули работают только по соглашениям интерфейса Супервизора, и в них не использованы никакие предположения относительно операционной системы, под которой функционирует Эуклидос, относительно конфигурации машины, диалога с оператором и т.д. Только Супервизор знает, как получить информацию от внешнего мира и как выдать туда результаты.

.1148. 2) Непосредственное общение между рядовыми модулями отсутствует. Любой из них может обратиться к другому, но только через Супервизор. Ни в одном из рядовых модулей не используются никакие предположения относительно того, будут ли другие рядовые модули резидентными или транзитными. Только Супервизор знает, какие части системы находятся в памяти, а какие надо вызывать с диска.

.1149. 3) Непременно резидентны (кроме Супервизора) только подпрограммы Резидента. В них сконцентрированы такие программы, которые часто нужны всем (или многим) модулям, но не зависят от внешнего мира и конфигурации Эуклидоса, а также неперемещаемые программы.

.1150. Итак – в Супервизоре сконцентрировано всё, что специфично для внешней среды, в Резиденте – всё остальное, что нужно всем.

.1151. Легко понять, что такая организация рассчитана на то, чтобы путем замены супервизоров легко менять конфигурацию (соотношение резидентных и транзитных частей) Эуклидоса и заставить его функционировать под разными операционными системами (или вообще без них) с разной конфигурацией машины.

.1152. Таким образом, в Эуклидосе может быть несколько различных супервизоров, которые, естественно, соблюдают единый интерфейс в общении с другими модулями.

.1153. Такова физическая организация Эуклидоса: он состоит из программных модулей, которые транслируются отдельно, два из них особенные и обязательно резидентны, а все остальные могут быть как резидентными, так и транзитными; только один из модулей зависит от операционной системы.

.1154. Логическая же организация Эуклидоса мало связана с физической. Она основывается на концепции виртуальных подпрограмм и будет описана немножко ниже.

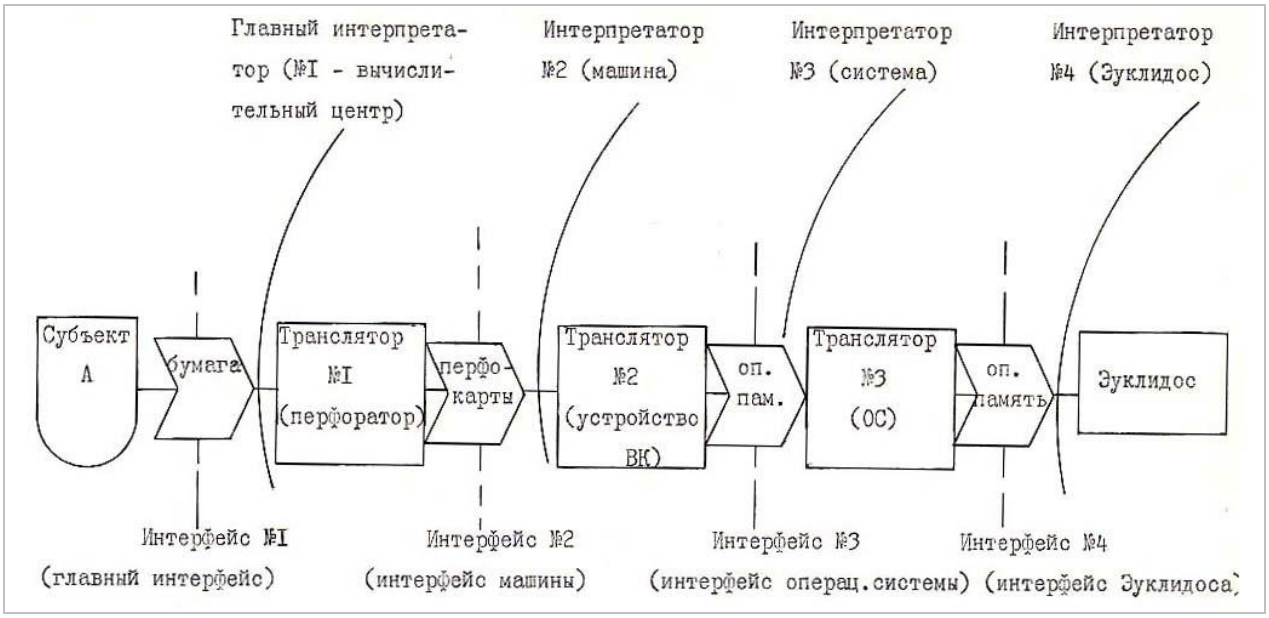
.1155. Все модули Эуклидоса носят такие названия, как EUKLIDA, EUKLIDB... и т.д., полученные комбинируя слово EUKLID с буквами латинского алфавита. Под такими именами они обычно хранятся на дисках. В описаниях же используются только последний (различительный) символ (или два символа), например, «модуль А» или его название на человеческом языке, например, «Супервизор». Эта одна (или две) различительные буквы называются идентификатором модуля.

.1156. Исходные тексты Эуклидоса представляют собой набор макрокоманд, хотя при генерации программ макросредства обычно не используются. Я отдавал предпочтение макрокомандам по сравнению с COPY текстами из-за особенностей Ассемблера/360: макрорасширения он выдает в выравненном на определенные позиции виде, как бы ни были отперфорированы макроопределения. Перфорировал же я обычно с минимальным числом пробелов.

.1157. Имена макрокоманд начинаются с букв EU, после чего идет буква-идентификатор модуля и далее какая-нибудь мнемоника или номер.

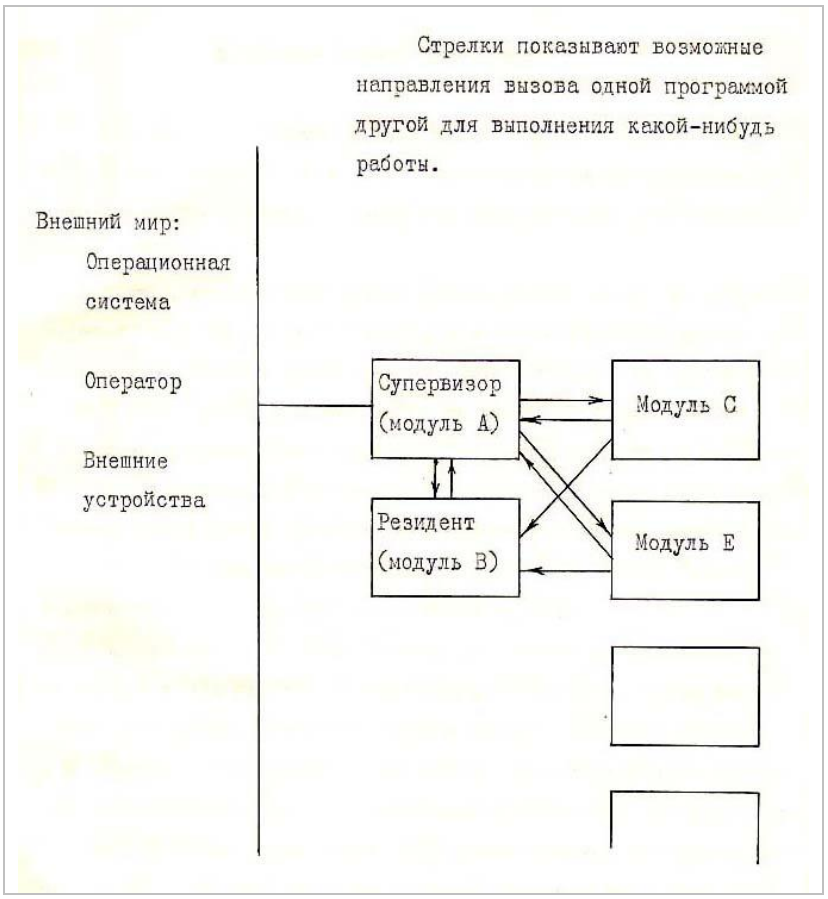
.1158. Все общее для разных модулей поля описаны в фиктивных секциях DSECT. Макрокоманды с этими секциями вызываются во всех трансляциях. Трансляция любого модуля начинается с макрокоманды MODULE, в операндах которой указывается, какой модуль транслируется. Эта макрокоманда вызывает все необходимые описания фиктивных секций, задает информацию о стандартных соглашениях по базированию (операторы USING) и генерирует управляющие карты для каталогизации модулей в библиотеках.

.1159.



Фиг.2. Иерархия интерпретаторов

.1160.



Фиг.3. Блок-схема Эуклидоса

8. Рабочая память Эуклидоса

1980.02

.1161. Как уже было сказано, один только Супервизор знает, когда, где, в каких адресах и в каком порядке расположены модули Эуклидоса и рабочие поля, то есть, всё это не важно для собственно системы Эуклидос.

.1162. Важно лишь то, что где-то (то ли после, то ли до модулей) имеется сплошной участок памяти, в котором Эуклидос запоминает теории (см. Фиг.4 {.1179}). Адрес первого байта этого поля хранится в ячейке ATHB, адрес первого байта за полем – в ячейке APAE.

.1163. Участки этого поля выдаются модулям в постоянное или временное пользование. В постоянное пользование может быть выдан участок любой длины. Эти участки выдаются с начала рабочей памяти, и первый свободный байт после навсегда занятой памяти идентифицируется адресом, хранящимся в ATHE.

.1164. Во временное пользование выдаются участки фиксированной длины и с конца памяти. Длина буфера – 64 байта. Когда модуль возвращает буфер, тот помещается в список свободных буферов и в следующий раз выдается другому модулю. Если модуль запросил динамическую память, а свободных буферов нет, то организуется новый буфер, и указатель APAB передвигается навстречу указателю ATHE. После некоторой пиковой нагрузки указатель APAB прекращает перемещаться.

.1165. Если указатель APAB оказался меньше указателя ATHE, то ресурсы памяти исчерпаны, и Эуклидос аварийно прекращает работу.

.1166. Управляют этой памятью четыре общие подпрограммы:

.1167. а) BETT – выдает постоянную память;

.1168. б) BETTA – аналогичный смысл и интерфейс, но адрес выделенного поля выравнен на границу слова, что может привести к потере 1–3 байтов;

.1169. в) ВЕТВОХ – выделить буфер динамической памяти;

.1170. г) ВОТВОХ – освободить буфер.

.1171. Все свои построения Эуклидос осуществляет в этой рабочей памяти, поэтому от ее величины существенно зависят возможности системы. Диски используются только для хранения библиотеки Эуклидоса. Рабочие файлы на дисках Эуклидос не создает.

.1172. Выделенные участки рабочей памяти (буферы) Эуклидос связывает в списки. В программистском мире известно много различных способов организации списков. В Эуклидосе используется следующий способ (см. Фиг.5 {.1180}):

.1173. Где-то выбираются два слова памяти, называемые началом или указателем списка. Первое слово содержит адрес первого буфера памяти. Первое слово этого буфера содержит адрес второго буфера и т.д.: каждый буфер содержит ссылку на следующий. Последний буфер списка ссылается «обратно» на указатель, что и служит признаком конца списка. Указатель пустого списка хранит свой собственный адрес.

.1174. Второе слово указателя списка содержит адрес последнего буфера списка, второе слово этого буфера – адрес предпоследнего и т.д.: ссылки идут в обратном порядке. Это позволяет найти последний буфер списка так же легко, как и первый, просматривать список в обратном направлении, легко вставлять и вынимать элементы из середины списка.

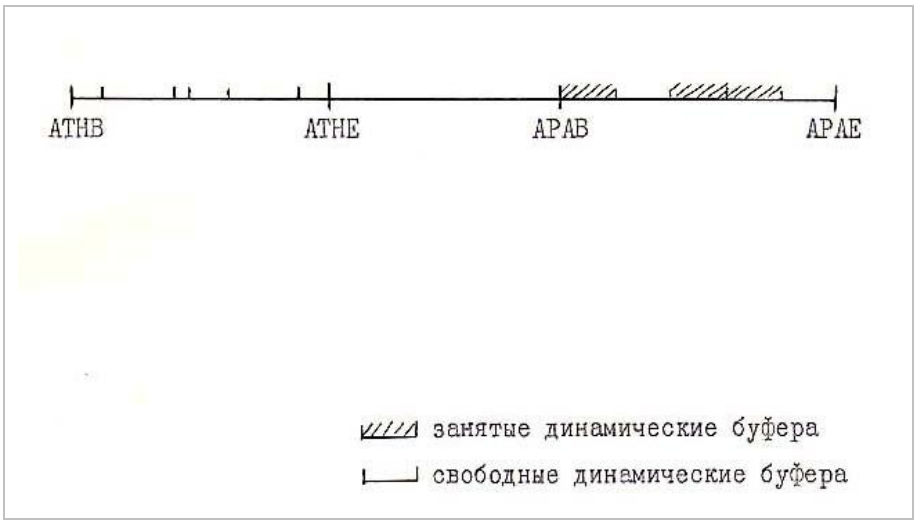
.1175. Мне эта организация кажется изящной тем, что все операции в этом списке проводятся абсолютно одинаково с первым, последним или средним элементом и независимо от того, пуст ли список, содержит ли один элемент или много. Но эта универсальность приводит к главному недостатку такой организации: при ошибке из списка в качестве элемента может быть выдан сам указатель списка. Тем не менее из-за логической простоты я продолжаю предпочитать эту организацию списка всем другим.

.1176. Списки позволяют просматривать буфера, разбросанные хаотично по всей памяти так же легко, как будто они расположены подряд (к тому же буфера могут быть различной длины) и, главное, позволяют строить структуры, длина которых заранее не известна, легко менять логические связи между ними.

.1177. Однако списки очень уязвимы, и малейшие нарушения в них выводят из строя всю систему, причем для ошибок в работе со списками (в отличие от других ошибок программиста) характерно, что списки моментально так перепутываются, что чрезвычайно трудно установить первопричину ошибки. Поэтому в Эуклидосе действует простая (но достаточно эффективная) система защиты от ошибок в работе со списками.

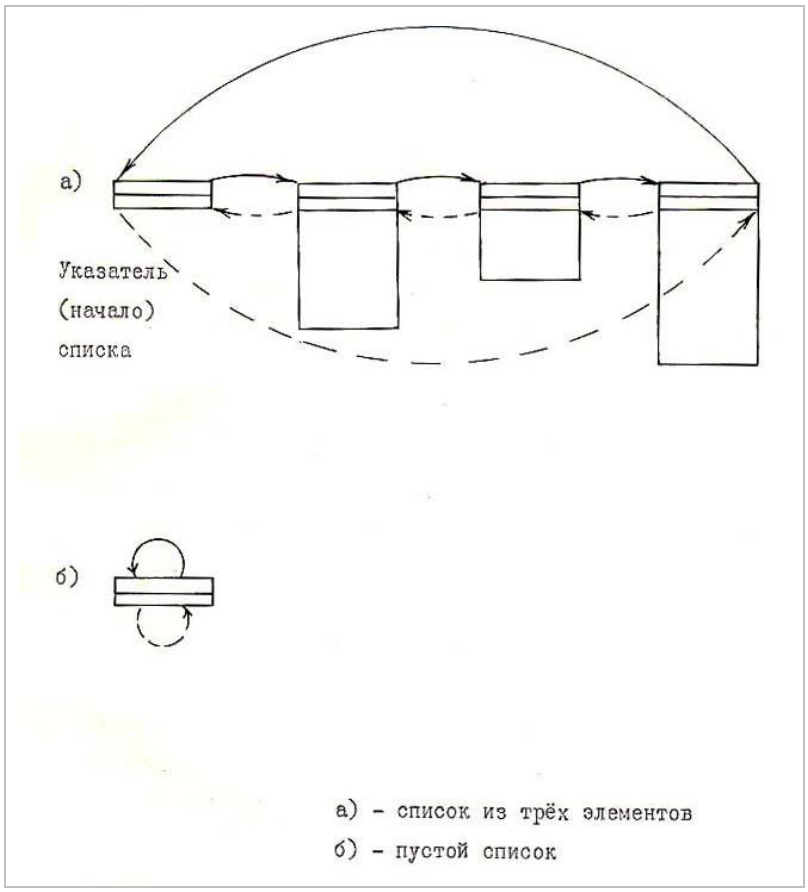
.1178. Помещение элемента в список и выборка из него осуществляются только специальными общими подпрограммами. Они при выдаче элемента из списка заносят в первые два слова элемента (вместо адресов связок) код, который невозможно интерпретировать как адрес, а при помещении в список контролируют наличие этого кода. Это позволяет сразу обнаружить помещение в список элемента повторно и помещение в список чего-то такого, что не предназначено для списка. Спасти от всех ошибок этот контроль, естественно, не может, но он исключает самую опасную и самую распространенную ошибку – повторное помещение в список.

.1179.



Фиг.4. Организация рабочей памяти

.1180.



Фиг.5. Организация списков

9. Свойства программ

1980.03
(через 1 месяц)

.1181. Рабочие модули Эуклидоса вызывают друг друга через Супервизор. Чтобы рассмотреть этот аппарат, нужно оговорить некоторые свойства программ. Не всякая программа обладает этими свойствами.

.1182. Повторновходимой называется такая программа, которую, однажды выполнив, можно запустить второй раз и получить правильные результаты. Повторновходимость программы обеспечивается тем, что она не модифицирует и не разрушает свое тело, а относительно своих внутренних полей не предполагает, что при входе они будут такими, какими заданы при трансляции. Все модули Эуклидоса должны быть повторновходимыми. Если модуль С вызывается модулем М после того, как его вызвал модуль Е и С еще сохранился в памяти, Супервизор имеет право не загрузить его заново с диска.

.1183. Переместимой называется такая программа, которая может быть загружена в любое место памяти и работать там правильно. Переместимость обеспечивается тем, что все адреса программа вычисляет только после загрузки, копия ее на диске абсолютных адресов не содержит. Все модули Эуклидоса должны быть переместимыми. Супервизор имеет право загрузить их в любое свободное место памяти.

.1184. Передвигаемой в какой-нибудь точке называется такая программа, выполнение которой можно в этой точке прервать, копию программы в памяти уничтожить, потом загрузить программу в другое место памяти, запустить с точки прерывания и получить правильный результат. Передвигаемость обеспечивается тем, что программа не использует своих внутренних полей, всю информацию держит в регистрах и чужих полях, а адреса своих частей (обычно это адреса возвратов из подпрограмм) держит не в виде абсолютных адресов, а в виде смещения

относительно начала программы. Те модули Эуклидоса, которые обращаются к другим модулям, должны быть передвигаемы в точках обращения к Супервизору для вызова другого модуля. Супервизор имеет право на время выполнения вызванного модуля вытеснить первый из памяти, а потом загрузить его в первое свободное место.

.1185. Реентерабельной в какой-нибудь точке называется такая программа, которую можно в этой точке прервать, запустить сначала, выполнить до конца, потом запустить с прерванного места и получить правильный результат. Реентерабельность обеспечивается тем, что программа не изменяет свое тело, не работает со своими внутренними полями, всю информацию держит в регистрах и в чужих полях. При наличии повторновходимости, обязательной для всех модулей Эуклидоса, реентерабельность оказывается более слабым свойством по сравнению с передвигаемостью, то есть, всякая передвигаемая программа автоматически реентерабельна, но не наоборот. Отсюда следует, что все рабочие модули Эуклидоса, вызывающие другие модули, реентерабельны и могут вызывать сами себя, или модуль Е, вызванный модулем С, может сам обратиться к модулю С. Такие обращения называются рекурсивными, а программы, допускающие такие вызовы – рекурсивными программами.

10. Логическая организация Эуклидоса

1980.04
(через 1 месяц)

.1186. Чтобы понять общую логическую структуру Эуклидоса, лучше всего начать с некоторого общеизвестного стандарта организации программ, к которому программисты обычно приходят в первые месяцы своей программистской карьеры.

.1187. Если программа не служит для обработки различных асинхронных сигналов, например, прерываний ввода-вывода (там применяются другие методы), то ее лучше всего организовать как показано в Фиг.6 {.1204}. Имеется некоторый головной участок, который реализует самую основную логику программы. Он для выполнения каких-нибудь определенных действий обращается к различным подпрограммам (по команде BAL). Каждая такая подпрограмма реализует основную логику выполняемого ею действия, а для осуществления еще более детализированных действий обращается к подпрограммам следующего уровня, и так далее до программ самого низкого уровня, которые уже не обращаются к другим программам. Каждая подпрограмма обязана сохранить все регистры головной подпрограммы (кроме оговоренных в соглашениях интерфейса и передающих ответ) и должна восстановить эти регистры перед возвратом управления.

.1188. Таким образом, такая программа состоит из инициатора (головного блока, который с точки зрения данной программы не вызывается никем, с которого вся работа начинается и кончается), из ряда единообразных подпрограмм и из области констант (константами я здесь называю всё, что не является машинными командами: поля, таблицы, указатели и т.д.). Инициатор и все подпрограммы достаточно невелики, чтобы их можно было обозреть целиком и видеть сразу всю их логику, переходы, манипуляции с регистрами и полями. Подпрограммы можно свободно перетасовывать и размещать в удобном для изучения листинга порядке, и это абсолютно не отражается на работу программы.

.1189. Специальные исследования в этой области показывают, что производительность труда хорошего и слабого программиста отличается примерно в двадцать (!) раз. Это совершенно немыслимый разрыв для других областей деятельности, например, для промышленности. Из своего опыта я могу сказать, что этот фантастический разрыв получается главным образом за счет методов организации программы. Неструктурную программу с запутанными переходами, во-первых, труднее написать, и пока слабый программист ломает голову, куда бы теперь ему перейти, хороший уже стандартным образом построил каскад вложенных подпрограмм. Но главный выигрыш получается при отладке: пока слабый программист роется в куче листингов в поисках того, где же он «портит» седьмой регистр, хороший уже ушел отдыхать, так как все регистры у него модифицируются в пределах одной страницы распечатки, а насчет остальных страниц имеется стопроцентная гарантия, что все регистры там восстанавливаются.

1994.06.01 12:42 среда
(через 14 лет, 2 месяца)

.1190. С предыдущим абзацем у автора связаны воспоминания о странном – или знаменательном? – совпадении: после того, как он вечером написал этот текст, буквально на следующий день ему пришлось потратить несколько часов в поисках очень каверзной и непонятной ошибки в своей программе. Когда же она была, наконец, обнаружена, то оказалось, что в одной из подпрограмм не все регистры восстанавливались (что с программами этого автора не случалось уже, пожалуй, годами ни до, ни после этого), и она портила СЕДЬМОЙ регистр).

1980.04
(раньше на 14 лет, 2 месяца)

.1191. Это азбука программирования, и о ней не стоило бы здесь говорить, если бы эта структура программы не являлась бы отправной точкой к организации Эуклидоса.

.1192. Программа, организованная указанным выше способом, обладает двумя свойствами, которые в дальнейшем меня особо будут интересовать:

.1193. а) подпрограммы можно свободно перемещать;

.1194. б) обращение к подпрограмме остается одним и тем же (команда BAL после подготовки параметров), где бы подпрограмма не находилась.

.1195. В реализации этих свойств нет никаких проблем, пока у Вас с десяток подпрограмм, все они транслируются вместе и находятся в одной фазе программы.

.1196. Но если количество операторов в Вашей программе будет стремительно приближаться к десяти тысячам, то жизнь заставит Вас разбить программу на ряд отдельно транслируемых модулей. Если теперь подпрограмма, находящаяся в модуле А, должна вызвать подпрограмму, находящуюся в модуле В, то соблюдение двух указанных выше свойств уже не так тривиально.

.1197. Если же объем Вашей программы будет измеряться десятками и сотнями килобайтов (без рабочих полей), то Вы захотите иметь транзитные фазы, вызываемые по необходимости. Теперь соблюдение указанных двух свойств становится еще сложнее, потому что нужной подпрограммы может вообще не быть в памяти.

.1198. При проектировании логической организации Эуклидоса я поставил перед собой задачу сделать так, чтобы:

.1199. а) Эуклидос состоял из отдельно транслируемых модулей;

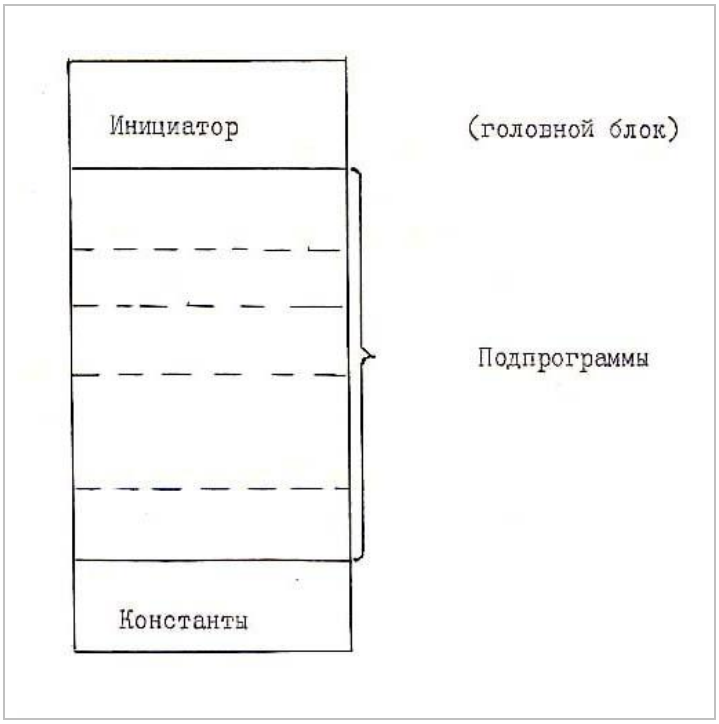
.1200. б) большинство этих модулей были транзитными, и их можно было бы загрузить в любое место памяти;

.1201. в) при этом подпрограммы можно было бы свободно перемещать из одного модуля в другой;

.1202. г) обращение к подпрограмме оставалось бы тем же (команда BAL) независимо от того, находится ли подпрограмма в том же модуле или в другом, в резидентном или транзитном.

.1203. Иными словами, Эуклидос – это программа, имеющая структуру, приведенную на схеме Фиг.6 {.1204}, но только виртуальную.

.1204.



Фиг.6. Типичная структура программы

11. Виртуальные подпрограммы

1980.04
(через 1 месяц)

.1205. Чтобы подпрограмма была доступна из любой точки Эуклидоса так, будто она находится в том же модуле, в системе встроены, так сказать, «удлинители» команды BAL. Все подпрограммы Эуклидоса с этой точки зрения можно подразделить на:

.1206. а) общие подпрограммы (для которых построен удлинитель и которые доступны из любой точки Эуклидоса в каком бы модуле они не находились);

.1207. б) частные подпрограммы (удлинители не построены, и подпрограмма доступна только из того модуля, где она находится).

.1208. В принципе можно построить удлинители для всех подпрограмм Эуклидоса, но реально в этом нет необходимости; достаточно иметь возможность легко создать удлинитель в случае необходимости.

.1209. С другой точки зрения все подпрограммы Эуклидоса можно подразделить на:

а) стандартные подпрограммы;

б) специальные подпрограммы.

.1210. Все стандартные подпрограммы реентерабельны, допускают рекурсивные вызовы; если они находятся в транзитных модулях, то передвигаемы. Области сохранения и рабочие поля им выделяются новые при каждом вызове (активации) в виде стандартного буфера, что и обеспечивает им указанные свойства. Все рабочие модули Эуклидоса включают только стандартные подпрограммы.

.1211. Специальные подпрограммы – это небольшая группа подпрограмм, которые не обладают указанными свойствами. Это, в первую очередь, подпрограммы, задействованные при вызове стандартных подпрограмм (выделение буфера и т.д.). Специальные подпрограммы имеются только в резидентных модулях (A и B), они никогда не обращаются к стандартным подпрограммам, но сами могут быть ими вызваны.

.1212. Как стандартные, так и специальные подпрограммы могут быть общими или частными.

.1213. Теперь рассмотрим механизм виртуального вызова подпрограмм. Когда в какой-нибудь точке Эуклидоса в модуле M выполняется команда «BAL 14,P», может произойти одно из трех событий:

.1214. а) подпрограмма P имеется в том же модуле M – произойдет самое тривиальное обращение;

.1215. б) P является специальной подпрограммой и находится в модулях A и B; при трансляции модуля M имя P было описано в фиктивной секции AMO или BMO, базируемой регистром 9 или 11, и фактический переход осуществится по этим регистрам и попадет на удлинитель – команду перехода на нужную подпрограмму;

.1216. в) P является стандартной подпрограммой в модуле C; при трансляции модуля M имя P было описано в фиктивной секции BMO, и фактически переход осуществляется по 11 регистру, где попадает на первую часть удлинителя, которая состоит из двух команд: загрузить в регистр 13 код подпрограммы и перейти к Супервизору. Код подпрограммы состоит из кода модуля (первый байт справа) и индекса подпрограммы в этом модуле (остальные байты регистра). Супервизор (вторая часть удлинителя) проверяет, есть ли модуль C в памяти, и, если его нет, то загружает, а потом передает управление по указанному индексу на начало модуля C, где отрабатывает третья часть удлинителя – команда перехода к нужной подпрограмме.

.1217. Таким образом, удлинители могут быть двух типов:

.1218. а) одна единственная команда перехода для специальных подпрограмм;

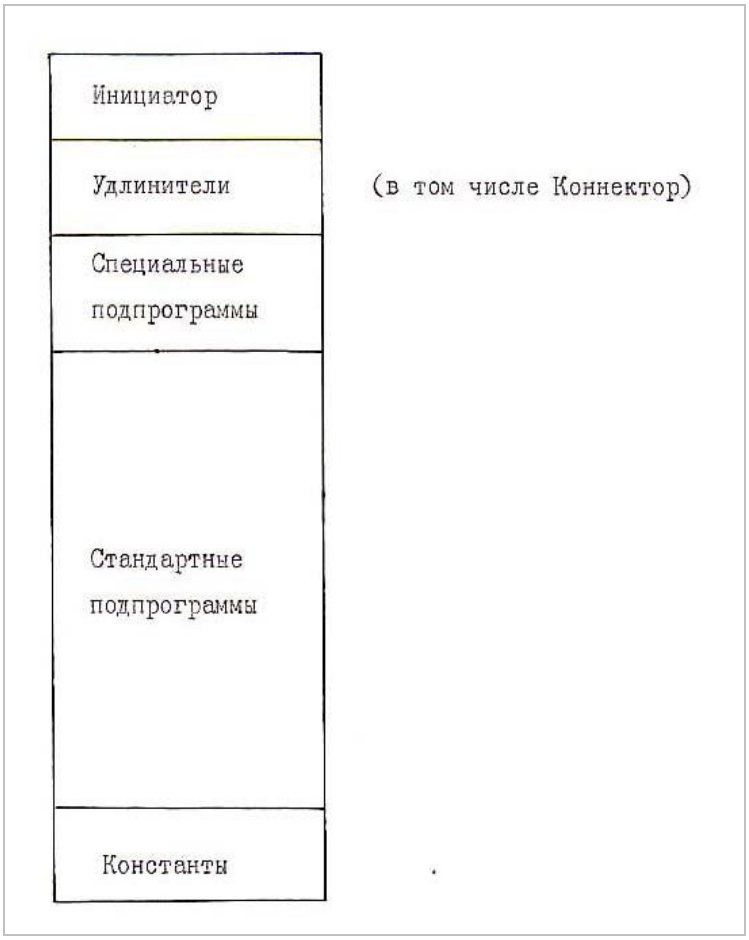
.1219. б) две команды в модуле B, одна команда в вызываемом модуле C и прохождение через блок Супервизора (Коннектор) – для стандартных подпрограмм.

.1220. Итак, логически Эуклидос имеет структуру, приведенную на схеме Фиг.7 {.1223}, которая образована из структуры Фиг.6 {.1204}. Эти логические части Эуклидоса распределяются по модулям, причем Инициатор всегда находится в Супервизоре, специальные подпрограммы – только в Супервизоре и Резиденте, а стандартные подпрограммы могут быть во всех модулях. В Супервизоре размещаются те специальные и стандартные подпрограммы, которые зависят от операционной системы; в Резиденте те специальные и стандартные подпрограммы, которые от операционной системы не зависят, но должны быть резидентными либо потому, что непередвигаемы, либо потому, что очень часто нужны.

.1221. Основную массу подпрограмм Эуклидоса составляют размещенные в рабочих модулях стандартные подпрограммы, которые все передвигаемы, реентерабельны, рекурсивны и не зависят от операционной системы. Размещение их по рабочим модулям имеет значение только с точки зрения оптимизации межмодульных обращений (минимизации частоты загрузки транзитных модулей) и с точки зрения чтения листингов. Иначе распределение подпрограмм по рабочим модулям безразлично, и может легко меняться. Перемещение подпрограммы из одного модуля в другой требует только перестройки удлинителей. Чтобы при таком переносе не появились повторяющиеся имена, метки должны быть уникальными не только в пределах модуля, но и по всей системе. Чтобы обеспечить уникальность имен, во время разработки Эуклидоса применялась специальная программа, распечатывающая в алфавитном порядке все встречающиеся в системе имена.

.1222. На самом деле нет необходимости, чтобы все подпрограммы рабочих модулей были передвигаемы и рекурсивны (это необходимо лишь для некоторых), но потери на приобретение этих свойств настолько незначительны, что я предпочел иметь единую организацию для всех этих подпрограмм.

.1223.



Фиг.7. Виртуальный Эуклидос

12. Регистры

1980.04

.1224. Итак, логически Эуклидос организован по принципу виртуальных подпрограмм, то есть, при написании очередной программы достаточно знать, что необходимая подпрограмма вообще имеется в системе, и можно дальше не заботиться о том, где она на самом деле находится: где бы она ни находилась, эффект будет такой же, как будто она размещена в том же модуле.

.1225. Аппарат виртуальных подпрограмм требует от них соблюдения некоторых правил или общесистемных соглашений по использованию регистров, полей и по интерфейсам.

.1226. Для стандартных (рекурсивных, реентерабельных и передвигаемых) подпрограмм эти правила таковы:

.1227. 1) Подпрограмма может использовать для своих целей регистры 0 по 8 (регистры 13 и 14 можно использовать краткосрочно): между обращениями к другим подпрограммам).

.1228. 2) Регистры 9 по 15 использует аппарат виртуальных подпрограмм; регистры 9 по 12 и регистр 15 подпрограмме запрещается модифицировать. Эти регистры используются следующим образом:

.1229. а) регистр 9 хранит базис Супервизора (модуля А); благодаря ему из любой точки Эуклидоса доступны описанные в секции АМО специальные подпрограммы Супервизора и общесистемные поля, размещенные в модуле А;

.1230. б) регистр 10 хранит базис рабочего модуля, то есть, базис самой работающей стандартной подпрограммы;

.1231. в) регистр 11 хранит базис Резидента (модуля В), благодаря ему из любой точки Эуклидоса доступны описанные в секции ВМО специальные подпрограммы Резидента и

общесистемные поля, находящиеся в модуле B, а также удлинители общих стандартных подпрограмм;

.1232. г) регистр 12 хранит базис рабочих полей работающей в данный момент подпрограммы; если она произведет обращение к другой подпрограмме, то здесь будут запомнены ее регистры; здесь имеются также четыре слова памяти, которые подпрограмма может использовать как рабочую область без особых на то приготовлений; если ей этой области мало, то она должна запрашивать буфер у BETBOX;

.1233. д) в регистре 13 вызванная стандартная подпрограмма может передать код возврата (обычно это индексы для таблицы переходов: 0, 4, 8 и т.д.); в общем – содержимое регистра 13 всегда передается из вызванной стандартной подпрограммы к вызвавшей;

.1234. е) в регистр 14 вызывающая подпрограмма при обращении к другой подпрограмме заносит адрес возврата; аппарат виртуальных вызовов вычитает из этого адреса базис программы (регистр 10) и запоминает как относительный адрес возврата. Когда управление возвращается данной подпрограмме, к этому относительному адресу базис программы прибавляется; если за время выполнения вызванной виртуальной подпрограммы (и вызванных, в свою очередь, ею виртуальных подпрограмм) модуль с головной программой пришлось вытеснить из памяти и при возврате управления снова загрузить в другую область, то новый базис программы будет отличаться от старого;

.1235. ж) регистр 15 хранит базис рабочих полей той подпрограммы, которая вызвала данную; благодаря ему подпрограмме доступны рабочие поля головной программы, и она может модифицировать запомненные там регистры головной программы.

.1236. 3) Первой командой стандартной подпрограммы обязательно должна быть команда «BAL 13,BOOK». Специальная подпрограмма BOOK запоминает регистры головной подпрограммы и относительный адрес возврата, выделяет новые рабочие поля вызванной подпрограмме и устанавливает в соответствии с новым положением регистры 12 и 15.

.1237. 4) Заканчиваться стандартная подпрограмма должна безусловным переходом на блок BACK, который освобождает ее рабочие поля и возвращает управление головной подпрограмме.

.1238. 5) При входе в подпрограмму ей передаются все регистры 0–8 головной программы; при выходе головная получает от подпрограммы только 13 регистр (и смодифицированные рабочие поля, как это оговорено соглашениями интерфейса).

.1239. При вызове специальных подпрограмм и при возврате из них аппарат виртуальных подпрограмм не задействован (они сами могут быть частью этого аппарата), поэтому правила интерфейса этих подпрограмм иные:

.1240. 1) Регистры головной программы они сохраняют и восстанавливают сами в фиксированной области сохранения; управление возвращают по абсолютному адресу в регистре 14.

.1241. 2) Код возврата обычно передается в виде признака результата.

.1242. 3) Они имеют право использовать кроме регистров 0–8 также и регистры 10, 12–15, разумеется, предварительно сохранив их (фактически даже и тот из регистров 9 и 11, который не служит ее базисом).

.1243. Благодаря этим общесистемным соглашениям ни стандартные, ни специальные подпрограммы вообще никогда не заботятся о своих базисных регистрах, о базисах вызываемых ими программ и рабочих полей.

.1244. Регистры 9 и 11 (базисы AMO и BMO) устанавливаются раз и навсегда Инициатором; регистр 10 – Коннектором при обнаружении резидентного или загрузке транзитного модуля, а регистры 12 и 15 – при входе и выходе из стандартной подпрограммы (устанавливаются специальными подпрограммами BOOK и BACK).

.1245. Принципиальную структуру Супервизора, Резидента и рабочего модуля можно увидеть на схемах Фиг. 8 {.1246}, 9 {.1247}, 10 {.1248}.

.1246.

Фиг.8. Структура Супервизора

.1247.

Фиг.9. Структура резидента

.1248.

Фиг.10. Структура рабочего модуля

13. Информационные потоки Эуклидоса

1980.04

.1249. Как всякая программа, Эуклидос обрабатывает какую-то поступающую к нему информацию и выдает какие-то результаты своей деятельности.

.1250. Всю доступную Эуклидосу входную информацию (ее лучше всего представлять себе как некоторое множество байтов) можно в первую очередь подразделить на две группы:

.1251. 1) Внешняя информация или ВТОК (входной поток, поступающий извне, от устройств типа ввода карт, дисплея и т.д.).

.1252. 2) Внутренняя информация или библиотека Эуклидоса. Она хранится на устройствах типа магнитных дисков (точное местонахождение библиотеки и различных ее частей зависит от операционной системы, конфигурации машины и известно одному только Супервизору).

.1253. По формату хранимых единиц библиотека Эуклидоса состоит из двух подбиблиотек (см. схему Фиг.11 {.1286}):

.1254. 1) программотека – множество байтов собственно программ Эуклидоса; это тексты на машинном языке. Физической единицей программотеки является модуль Эуклидоса (Эуклидос подвергает анализу свои модули, загруженные в память, поэтому как входная информация Эуклидоса программотека состоит из абсолютных модулей).

.1255. 2) либротека – множество байтов текстов на немашинных языках (на Эуклидоле, Ассемблере и человеческом языке). Физической единицей либротеки является карта – 80 байтов информации. Сообразно языкам, на которых написаны тексты либротеки, она делится на три главных раздела:

.1256. а) эуклидотека – хранит тексты на Эуклидоле;

.1257. б) ассемблотека – хранит тексты на Ассемблере (исходные программы Эуклидоса);

.1258. в) графотека – хранит тексты на человеческом языке;

.1259. г) список разделов либротеки может быть в дальнейшем расширен.

.1260. С другой точки зрения (функциональной) всю доступную Эуклидосу информацию можно подразделить на две главные группы:

.1261. 1) Главный входной поток – та информация, для обработки которой, собственно, Эуклидос и создан. Это вток и эуклидотека – тексты на Эуклидоле, подвергаемые им обработке по своему главному назначению. Считается, что за одно выполнение Эуклидос обрабатывает одну теорию.

.1262. 2) Информация для автоскриптов (самоописаний) Эуклидоса – это программотека, ассемблотека и графотека. Для составления автоскриптов Эуклидос подвергает обработке собственные модули программ, собственные исходные тексты на Ассемблере и тексты описаний на человеческом языке, хранящиеся в графотеке.

.1263. 3) Список функциональных групп информации может быть в дальнейшем расширен.

.1264. Таким образом, ассемблеротека и графотека с точки зрения Эуклидоса имеют одинаковое назначение и строение (формат перфокарт и т.д.) и объединяются в особый раздел библиотеки – скриптотеку, хранилище описаний Эуклидоса (что же такое текст программы на Ассемблере, если не самое точное ее описание?). С этой точки зрения ассемблеротека – это лишь тот отдел скриптотеки, который Эуклидос использует совместно с Ассемблером, а графотека – тот отдел, которым Эуклидос пользуется один.

.1265. В той мере, в какой Эуклидос черпает информацию для автоскриптов непосредственно из модулей своих программ и из их исходных текстов, эти описания в принципе никогда не могут устареть. Достаточно изменить саму программу, и Эуклидос автоматически будет по-новому описывать себя. Тексты графотеки и комментарии в программах (которые тоже используются Автодескриптором), конечно, могут устареть, но они обладают такой же модульностью, как и программы: если Вы имеете описание на 200 страниц, отпечатанное на машинке или в типографии, и одна страница уже устарела, то обычно у Вас нет возможности заменить эту одну страницу и спасти остальные 199; в графотеке же это можно сделать легко. Всегда легче следить, чтобы комментарии соответствовали программе, чем где-то отдельно накапливать материал для описания очередной версии. Кроме того, графотека на диске обычно перемещается вместе с самой программотекой, что исключает возможность того, что у Вас описания одной версии, а сама система – другой.

.1266. Физической единицей программотеки является модуль, скриптотеки – карта. Логическими единицами являются в программотеке блок и модуль (одновременно логическая и физическая единица), а в скриптотеке – скрипт и журнал.

.1267. Блок всегда целиком входит в какой-нибудь один модуль (не может быть блока, размещенного в нескольких модулях). Блоки между собой не пересекаются, но не всякий байт модуля принадлежит какому-нибудь блоку: в принципе могут быть байты модуля, не принадлежащие никакому блоку. Бывают блоки разных типов, например, подпрограммы, таблицы и т.д. (подробнее о типах блоков см. в автоскриптах Эуклидоса).

.1268. Журнал – крупная логическая единица скриптотеки. Каждый байт (и каждая карта) скриптотеки входит в один и только один журнал. Каждый журнал входит либо в ассемблотеку, либо в графотеку (нет журналов, пересекающих оба раздела скриптотеки).

.1269. Скрипт – основная логическая единица скриптотеки. Каждый скрипт входит в один и только один журнал, но не каждая карта журнала входит в какой-нибудь скрипт: могут быть карты журнала, не входящие ни в один скрипт. Скрипты не пересекаются. Бывают скрипты различных типов: исходные тексты программ, структуры, описания программ и т.д. (список типов см. в автоскриптах Эуклидоса).

.1270. Каждый блок и каждый скрипт входит в тот или иной (и только один) аппарат Эуклидоса. Аппарат – это совокупность программ и блоков (вместе с их описаниями), предназначенных для выполнения определенной работы. Каждый аппарат имеет свое мнемоническое обозначение. Так, например, аппарат дескрипт – это аппарат самоописания Эуклидоса; аппарат предикат – это средства выделения высказываний на Эуклидоле и т.д.

.1271. Аппараты делятся на рабочие (осуществляющие те функции, для которых Эуклидос создан) и вспомогательные. Перечни и точные описания аппаратов см. в автоскриптах Эуклидоса.

.1272. В эуклидотеке имеется только одна логическая единица – книга. Физической единицей информации как в эуклидотеке, так и во втоке служит перфокарта.

.1273. В каких отношениях эуклидотека, ассемблотека, графотека и программотека находятся с библиотеками и файлами операционной системы, знает только Супервизор, и это не имеет значения для остального Эуклидоса, который карты и модули этих хранилищ (как и карты втока) получает от Супервизора как три входные потока (см. Фиг.12 {.1283}):

.1274. а) эуклидоток, формируемый из втока со вставленными книгами эуклидотеки;

.1275. б) скриптоток, поступающий из скриптотеки;

.1276. в) программоток, постоупающий из программотеки.

.1277. То, какие источники, книги, журналы или модули будут вливаться в эти три потока, определяется состоянием пяти коммутаторов. Первые три из этих коммутаторов управляются пользователем, остальные два – самим Эуклидосом.

.1278. Коммутатор №1 определяет, какой из источников будет давать информацию во вток. Источниками могут быть, например, ввод карт, дисплей, пишущая машинка и т.д. Конкретный состав источников см. в автоскриптах Эуклидоса.

.1279. Коммутатор №2 определяет, будет ли эуклидоток поступать от втока или из эуклидотеки.

.1280. Коммутатор №3 определяет, какая из книг эуклидотеки будет поступать в эуклидоток.

.1281. Коммутатор №4 определяет, какой из журналов скриптотеки будет поступать в скриптоток.

.1282. Коммутатор №5 определяет, какой модуль Эуклидоса поступит в программоток.

.1283. Средства, имеющиеся у пользователя для управления коммутаторами №№ 1–3, см. в автоскриптах Эуклидоса.

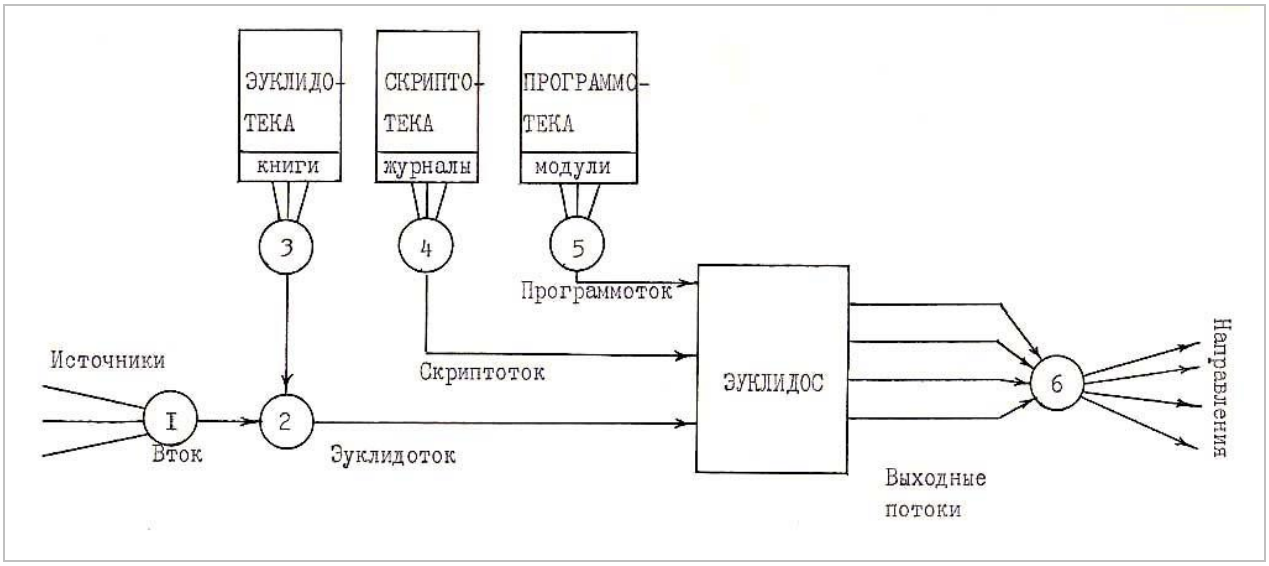
.1284. Выходную информацию Эуклидос выдает в ряде выходных потоков. Каждый выходной поток при помощи коммутатора N6 может быть направлен в одном из ряда направлений. Направлениями могут быть, например, консоль, принтер и т.д. Каждый выходной поток несет информацию определенного типа. Конкретный список выходных потоков, направлений и средства управления коммутатором N6 см. в автоскриптах Эуклидоса.

.1285. В один из выходных потоков поступают сами автоскрипты. Логической единицей автоскриптов является дескрипт. Это отредактированный и обработанный текст одного скрипта, пополненный сгенерированной Автодескриптором точной информацией. Автоскрипт – это множество дескриптов. Заказывая автоскрипт у Эуклидоса, можно указать разные комбинации дескриптов, например: всё, что относится к одной программе (ее описание, интерфейс, текст и т.д.) или только все интерфейсы всех программ, или только тексты и т.д. Средства управления Автодескриптором см. в автоскриптах, описывающих аппарат ДЕСКРИПТ.

.1286.

Фиг.11. Входная информация Эуклидоса

.1287.



Фиг.12. Информационные потоки

14. Резюме ЭУКЛИДОСА

§11. Резюме

1980.04

.1288. В этой медитации были описаны общие концепции и понятия, лежащие в основе идеологии системы Эуклидос, необходимые для дальнейшего, более подробного знакомства с этой системой программ.

.1289. Сам же Эуклидос был создан как реализация на ЭВМ транслятора и интерпретатора для языка точного описания теорий – Эуклидола. В дальнейшем описание языка ведется параллельно с описанием соответствующего аппарата Эуклидоса. Это необходимо в первую очередь для более глубокого понимания смысла и сущности конструкций языка. Описания же отдельных аппаратов были бы непонятны без знакомства с общими концепциями, изложенными здесь.

.1290. Сам же язык описания теорий Эуклидол (главная цель всего цикла медитаций) является алгоритмическим языком. Это вытекает из концепции отражения, согласно которой сущность теорий состоит в том, что они представляют собой алгоритмы, по которым человек в процессе отражения обрабатывает информацию об окружающем мире.

.1291. В самых основных чертах деятельность Эуклидоса и человека при дешифровке текстов языка совпадает. В своей сущности язык Эуклидол выполняет те же функции, что и человеческие языки, и отличается от них только тем, что его правила гораздо более систематизированы, их число минимизировано, и все они строго оговорены. Деятельность Эуклидоса представляет собой очень упрощенную, систематизированную, но в принципе своем адекватную модель человеческого мышления. Люди в своих головах делают примерно то же самое, что и аппараты Эуклидоса, но более окольными путями, менее организованно и целеустремленно, более щедро растрачивая средства.

.1292. Я не нашел лучшего способа рассказать о том, что из себя на мой взгляд представляет эта деятельность, чем показать это на ЭВМ.

§12. Послесловие при помещении в Ведду

1994.06.01 13:38 среда
(через 14 лет, 2 месяца)

.1293. Медитация ЭУКЛИДОС во время ее написания выполняла несколько функций:

.1294. 1) для «посторонних» читателей (математиков и т.п.) она должна была дать какое-то общее представление об организации разрабатываемой программной системы, не рассчитывая, конечно, что этот контингент читателей станет углубляться в детали {.1134};

.1295. 2) в то же время эти документы служили руководством для самого автора, ибо невозможно создавать большие системы, не фиксируя нигде своих технических решений (так как программы подготавливались на перфокартах, то даже возможности комментирования программ были крайне ограничены); эта (и следующие) медитация, таким образом, выполняла роль авторской документации, создаваемой параллельно с самой системой – иногда в качестве предокументации, до написания программ, иногда «по горячим следам» за самой программой;

.1296. 3) кроме того, автор надеялся, что «публикация» сборника «О природе чисел» поможет ему найти помощников в его работе {.2797}, и тогда эта (и следующие) медитации послужат в качестве источника информации для таких помощников.

.1297. Создавать же три разных документации для этих трех разных целей было немыслимо, т.к. автор и без того находился в катастрофическом цейтноте. Работа не была никем санкционирована и официально никем не поддерживалась, проводилась почти что подпольно. Начальство, как это описано в {.46} и других местах, о работе знало, но, тем не менее, не могло обеспечить достаточное количество времени и неограниченные возможности. Поэтому на автора давило ощущение необходимости быстрее создать что-то такое, что можно показать, за что зацепиться, чтобы как-то получить возможность продолжить работу (следствия такой атмосферы чувствуются во многих местах сборника «О природе чисел»).

.1298. Эуклидос создавался в промежутке между версиями 2.0 и 3.0 операционной системы «Диспетчер» этого же автора. Поэтому в организации Эуклидоса использовались (с усовершенствованиями) многие идеи Диспетчера-2 и, в свою очередь, многие идеи Эуклидоса были потом

воплощены в Диспетчер-3, операционную систему, которая весьма успешно функционировала все 1980-ые годы. Свою логическую вершину же эти идеи достигли в системе HUSK, которую автор сделал в 1990 году, и которая была загублена незавершенной...

.1299. Эуклидос был работоспособной программой, но он так и не был закончен по нескольким причинам:

.1300. 1) Непосредственной причиной прекращения работы над Эуклидосом был приказ начальства разрабатывать Диспетчер-3. Однако еще в течение нескольких следующих лет автор надеялся вернуться к Эуклидосу.

.1301. 2) Причиной, по которой он всё же не вернулся к этой системе, наряду с постоянной крайней занятостью, была реакция латвийских математиков на сборник «О природе чисел» и на дальнейшие («Преобразование», «Канториана»). Фундаментальные идеи, изложенные в этих сборниках, не были поняты и признаны математиками, но сначала существовала надежда, что эти идеи все-таки могут быть поняты и признаны, если им уделить достаточно внимания и рассмотреть их подробнее в дискуссиях. Поэтому автор дискуссиям отдавал предпочтение перед дальнейшей разработкой Эуклидоса (Эуклидос ведь нужен был не сам по себе, а только как иллюстрация к тем идеям; что толку иметь программу, если не признается тот фундамент, на котором она построена?; просто программ, чем-то манипулирующих, в мире тысячи, если не миллионы, и кого удивишь еще одной?). Таким образом, в ближайшие годы после прекращения работы над Эуклидосом, автор всё время, высвобожденное им для этой области деятельности, тратил на дискуссии с математиками, и результатом этих дискуссий являются сборники «Преобразование» и «Канториана».

.1302. 3) Однако дискуссии эти кончились крайне деструктивным, бессовестным, озлобленным и издевательским отношением латвийских математиков к автору, и после разрыва с ними уже тем более не было никакого смысла возвращаться к Эуклидосу. Более того, была прекращена вообще всякая деятельность автора в данной области.

.1303. Тем не менее, Эуклидос сохранялся в полуготовом состоянии на дисках вплоть до осени 1991 года, когда он при ликвидации ЕС ЭВМ был уничтожен руководством Института электроники вместе с Диспетчером-3, Хаскными системами и всеми другими прежними разработками автора.

.1304. То, что работы над Эуклидосом были в свое время остановлены, было в принципе, с сегодняшней точки зрения, хорошо – чем больше труда было бы в него вложено, тем больше потом было бы загублено. Теперь очевидно, что тогдашний Эуклидос, несмотря на блестящие идеи и технические решения, вложенные в него, не мог «выиграть сражение». Как продукт на ЕС ЭВМ (IBM/360, /370), он не мог получить массовое распространение, стать общедоступным для всех интересующихся. Технический уровень был низок (подготовка программ была возможна только на перфокартах, полностью отсутствовали машинные документаторы, и т.п.), и это, конечно, отразилось бы на качестве программной системы, какими бы виртуозными уловками автор ни старался бы преодолеть эти препятствия, изобретая «автоскрипты» (прообраз теперешних «хелпов», которых тогда еще в Латвии никто и не видел) и подобные вещи.

.1305. Теперь (1994 год) положение радикально изменилось. Массовое распространение получили персональные компьютеры, и Эуклидос, созданный для ЭТИХ машин, мог бы стать общедоступным и легко передаваться от одного заинтересованного лица к другому, а сохранность продукта уже не зависела бы от каких-то решений руководства какого-то института. Более того, распространение персональных компьютеров и происшедшая в СССР революция, приведшая к свободе печати, открыли возможность публиковать мои работы вопреки всем тем препятствиям, которые были на моем пути созданы латвийскими математиками и которые раньше казались непреодолимыми.

.1306. Уже в ноябре 1992 года было принято принципиальное решение о создании Ведды – Компьютерного архива документов Валдиса Эгле, – документов, которые могут быть распространены без помощи типографий – одним только компьютерным путем. Было решено в рамках Ведды опубликовать наряду с многими другими материалами также и полностью все документы, показывающие, какие идеи автор выдвигал уже в конце 1970-х и начале 1980-х годов и с какой тупостью и ограниченностью они были встречены латвийскими математиками, как эти математики загубили работу автора – ПОЧТИ загубили, ибо нынешнее возрождение этой работы происходит не благодаря, а вопреки и назло математикам.

.1307. Весной 1994 года было принято принципиальное решение не только опубликовать все «математические» документы автора, но и воссоздать Эуклидос на персональных компь-

ютерах. Так что, несмотря ни на что – Эуклидос идет, Эуклидос возвращается! Но то будет новая система[2] с новой документацией, в новой среде, с новыми возможностями. А эти медитации навсегда сохранятся в нетронутом виде как памятник той системе, которой не дали родиться и потом неродившуюся загубили.

[2] **Примечание 2008-го года**. В 1990-х годах я немножко предпринял попытку воссоздать Эуклидос на персональных компьютерах. Но далеко эти попытки не пошли. Нет, конечно, абсолютно никаких проблем (кроме труда и времени) в создании реальной, работающей программы (системы) Эуклидоса. Только смысла в этом нет. Что изменится, если я потрачу год своей жизни и создам эту программную систему, как создал многие другие до нее? Математики тогда поверят в правильность Веданской теории и поймут действительные основания математики? – Да ни черта они не поймут! Чтобы можно было понять, достаточно того, что она описана (как проект), а реализация ее ничего к этому не добавит. Только год моей жизни будет напрасно истрачен. Такие мысли и остановили меня в 1990-х годах. Для умных людей достаточно того, что УЖЕ написано, а дуракам ничто не поможет – в том числе реализованный и работающий Эуклидос.

8. Тетрадь PREDI

ПРЕДИКАТ
Основы языка Эуклидола

Не буди того, что отмечталось,
Не волнуй того, что не сбылось, –
Слишком раннюю утрату и усталость
Испытать мне в жизни привелось.

Сергей Есенин

Написано: 1980.02 – 1980.05 Рига

Медия PREDI (в Третьей Медиотеке медитация ПРЕДИКАТ) содержит общее описание проекта основ языка Эуклидола, предназначенного для компьютерно формализованного описания логических и математических объектов.

1. Тексты в Эуклидосе

1980.02
(раньше на 14 лет, 4 месяца)

.1308. В этой медитации будет описан один из аппаратов Эуклидоса – аппарат, носящий название ПРЕДИКАТ. Это служебный (вспомогательный) аппарат, предназначенный для того, чтобы выделять из текста, написанного на Эуклидоле, различные синтаксические единицы. Следовательно, с точки зрения языка здесь описаны правила построения высказываний на этом языке безотносительно к их смыслу или содержанию.

.1309. Но, прежде чем заняться разбором текстов Эуклидола, не помешает взглянуть на сами тексты с общей точки зрения.

.1310. В медитации ЭУКЛИДОС о текстах уже было сказано следующее:

.1311. а) что текстом называется множество знаков {.1010};

.1312. б) что тексты обязательно существуют в каком-нибудь интерфейсе {.1088};

.1313. в) что тексты по отношению к какому-нибудь интерпретатору делятся на данные и программы {.1089}.

.1314. Для алгоритмов работы с множествами может быть существенным или безразличным порядок элементов во множестве {.700}. В первом случае множество называется Порядком.

.1315. Для алгоритмов обработки текстов также может быть важен или безразличен порядок знаков, то есть, множество знаков может рассматриваться как линейный текст (линейная последовательность знаков) или нелинейный. Все тексты Эуклидола, как в главном интерфейсе, так и во всех внутренних интерфейсах Эуклидоса всегда линейны. Более того, все тексты внутренних программ Эуклидоса также линейны, то есть, в этих медитациях мы вообще будем иметь дело только с линейными текстами. Тем не менее, не помешает помнить, что линейные тексты – хоть и чрезвычайно важный, но всё же лишь частный случай текстов.

.1316. Вместе с каждым конкретным текстом, состоящим из материальных знаков (конкретное множество) можно рассматривать некий абстрактный текст (абстрактное множество), состоящий из позиций знаков; его можно представлять себе как ряд пустых клеточек, куда можно вписать тот или иной (но один) знак. В принципе могут быть тексты, где, если Вы вписали определенный знак в одну клеточку, то уже в соседнюю (или вообще какую-нибудь другую клеточку) можете вписать не любой знак, или же само деление на клетки изменилось. Если же наличие любого знака в любой клетке никак не влияет на возможности

помещения знаков во все остальные клетки, то такой текст называется свободным. Мы будем иметь дело только со свободными текстами, но не помешает помнить, что это опять-таки очень важный, но лишь частный случай текстов.

.1317. Если во все клетки можно вписать одинаковый набор знаков, то есть, если все клетки с этой точки зрения одинаковы (что в общем случае вовсе не обязательно), то текст называется ОДНОРОДНЫМ. Здесь мы будем иметь дело только с однородными текстами.

.1318. Однородный свободный линейный текст называется комбинационным. Итак, во всей системе Эуклидола и Эуклидоса имеются только комбинационные тексты, в которых:

.1319. а) определен порядок знаков;

.1320. б) все позиции могут содержать знаки из одного и того же набора Букв;

.1321. в) наличие знаков в одной позиции не влияет на знаки в других позициях.

.1322. Любой знак текста принадлежит к одному определенному множеству (букве), а множество всех букв называется алфавитом {.605}. Если все языки, фигурирующие в интерфейсах Эуклидоса, были похожи друг на друга тем, что все они – комбинационные, то по части алфавита языки в разных интерфейсах уже расходятся. Алфавит главного интерфейса (Эуклидол на бумаге) состоит из 76 букв; алфавит интерфейса №2 (на перфокартах) – из 256 букв, а алфавиты дальнейших (внутримашинных) интерфейсов – из двух букв (0 и 1).

.1323. Знак является Точкой теории текстов {.510}, то есть, он никогда не рассматривается как множество, которое само может иметь какую-то структуру. Разумеется, что на самом деле любой язык – множество (например, письменный знак – множество молекул красителя, колонка перфокарты – множество пробивок, бит оперативной памяти – множество ориентированных магнитным полем кристалликов и т.д.). Но, чтобы построить какую-нибудь конкретную теорию текстов, мы должны остановить свой спуск ко всё более элементарным единицам и сказать: «для данной теории вот эти объекты абсолютно элементарны, это точки – множества нулевого уровня». Конечно, не помешает при этом понимать, что эта фиксация уровня точек довольно произвольна, и всегда может возникнуть вопрос: «а почему, собственно, эти объекты, а не другие, более крупные или более мелкие, считать точками?».

.1324. Язык на перфокартах в интерфейсе №2 (см. {.1138}) меня не будет интересовать, и в дальнейшем мы будем иметь дело только с языком главного интерфейса (канонический Эуклидол) с 76-буквенным алфавитом и с языком внутримашинных интерфейсов с 2-буквенным алфавитом.

.1325. Читатель, возможно, уже заметил, что в этой главе получили конкретное применение некоторые вещи, о которых я в общем виде говорил уже в медитации ТЕОРИКА {.410} (точки, уровни множеств, отношения знаков, букв, алфавита). Таким образом, эти размышления заодно служат и примером применения идей теорики. Здесь я теорию текстов Эуклидола изложу с теоретической точностью (то есть, в терминах теории множеств, но без формализованного языка). Как только мы дойдем до соответствующего аппарата Эуклидола, эти же рассуждения послужат мне одним из первых примеров применения языка фундаментальной точности: я повторю всё это на Эуклидоле.

2. Классификация структур

1980.03
(через 1 месяц)

.1326. Возьмем произвольное множество знаков и назовем его структурой. Возьмем произвольную Группировку {.679} этой структуры и части ее назовем записями.

.1327. В этой главе мы изучим возможные взаимоотношения структуры с ее записями, чтобы в дальнейшем, встретив какую-нибудь структуру, смогли легко ее охарактеризовать. Иными словами, задача заключается в том, чтобы построить Классификацию структур, основанную на взаимоотношениях структуры с ее записями.

.1328. Сначала введем одно понятие, основанное на определенное в линейных текстах отношение следования знаков.

.1329. Любое непустое множество подряд идущих знаков называется отрезком. Отрезками являются, например, байты и слова в памяти машины или то, что мы понимаем под словом, когда читаем книгу. Отрезками являются также объединение нескольких байтов (и вообще любой

непрерывный участок памяти), фразы текста (вместе с пробелами) и т.д. Однако отрезками не является объединение двух байтов, находящихся в разных местах памяти и т.д.

.1330. Отрезки являются множествами первого уровня или Фигурами (всякий отрезок – фигура, но не всякая фигура – отрезок).

.1331. Всякое построение классификации заключается во многократном делении исходного множества различными секаторами {.684}. Для классификации множества структур сначала выберем следующие два секатора:

.1332. а) множество СА всех структур, являющихся отрезком;

.1333. б) множество СВ всех тех структур, все записи которых являются отрезками.

.1334. Эти два секатора должны давать четыре таксона первой степени в нашей классификации, но один из этих таксонов всегда пуст (множество тех структур, которые сами являются отрезками, но хотя бы одна запись в них не является отрезком). Эта деформация таксонов вызвана тем, что секатор СВ включает секатор СА. В общем виде это было рассмотрено в {.681}.

.1335. Итак, эти два вложенных секатора дают нам три таксона первой степени в классификации структур:

.1336. а) массивы – структуры типа А, являющиеся отрезками; это множества подряд идущих знаков, все их записи также – отрезки;

.1337. б) файлы – структуры типа В, которые сами не являются отрезками, но все записи которых – отрезки; записи могут быть разбросаны по тексту в разных местах, но каждая запись – непрерывный участок текста;

.1338. в) облака – структуры типа С, которые и сами не отрезки, и хотя бы одна запись в них не является отрезком.

.1339. Таксоны второй степени образуем при помощи еще двух секаторов:

.1340. а) множество СЕ структур, в которых число записей известно к началу их чтения;

.1341. б) множество СН структур, в которых число записей оговорено в предварительных соглашениях по языку.

.1342. Опять у нас секатор СН входит в секатор СЕ, поэтому новые секаторы разобьют каждый таксон первой степени на три, а не на четыре части:

.1343. а) структуры типа Х фиксированной длины – число записей оговорено в предварительных соглашениях;

.1344. б) структуры типа Р переменной длины – число записей не оговорено в предварительных соглашениях, но известно к началу чтения (то есть, оговорено в динамических соглашениях);

.1345. в) структуры типа О неопределенной длины – число записей к началу чтения не известно, должен быть оговорен признак конца структуры.

.1346. Итак, мы имеем девять таксонов второй степени.

.1347. Таксоны третьей степени образуем еще двумя секаторами:

.1348. а) множество СМ тех структур, максимальное число записей которых оговорено к началу создания структуры;

.1349. б) множество СК тех структур, в которых это число оговорено в предварительных соглашениях.

.1350. На этот раз не только СК входит в СМ, но и СН входит в СК (см. Фиг.6 {.1375}). Поэтому мы имеем в каждом таксоне первой степени не 12 и не 9, а всего 7 таксонов третьей степени:

.1351. а) структуры фиксированной длины;

.1352. б) структуры переменной длины с предварительно оговоренной максимальной длиной;

.1353. в) структуры переменной длины с динамически оговоренной максимальной длиной;

.1354. г) структуры переменной длины с неоговоренной максимальной длиной;

.1355. д) структуры неопределенной длины с предварительно оговоренной максимальной длиной;

.1356. е) структуры неопределенной длины с динамически оговоренной максимальной длиной;

.1357. ж) структуры неопределенной длины с неоговоренной максимальной длиной.

.1358. Всего имеем 21 таксон третьей степени.

.1359. С точки зрения перехода от таксонов второй степени к таксонам третьей степени это выглядит так, что каждый таксон второй степени делится на три таксона:

.1360. а) структуры типа М оговоренной (в предварительных соглашениях) длины;

.1361. б) структуры типа К ограниченной (динамическими соглашениями) длины;

.1362. в) структуры типа Т неограниченной длины.

.1363. Структуры типа Х могут иметь только тип М.

.1364. Понятно, что каждая запись может также быть структурой и принадлежать тому или иному таксону этой классификации по отношению к своим записям.

.1365. Четвертую степень таксонов построим, основываясь на длине записей и используя еще два секатора:

.1366. а) множество СО структур, у которых записи имеют всегда одинаковую длину;

.1367. б) множество СР структур, у которых эта длина оговорена предварительно.

.1368. Секатор СР входит в СО. Эти секаторы дают 3 таксона четвертой степени в каждом таксоне третьей степени:

.1369. а) однородные структуры типа Е, у которых все записи фиксированной длины;

.1370. б) гомогенные структуры типа Н, у которых все записи переменной длины, эта длина оговорена динамически, но один раз для всех записей;

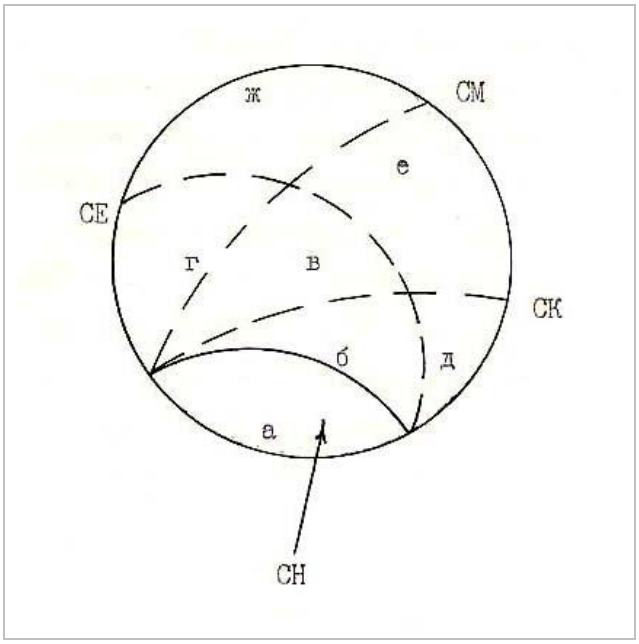
.1371. в) неоднородные структуры типа I, в которых имеются записи неопределенной длины или записи переменной длины, длина которых оговаривается отдельно для каждой записи.

.1372. Такова та классификация структур, которую я в дальнейшем буду использовать. При ее построении было задействовано 8 секаторов; она состоит из трех таксонов первой степени, 9 таксонов второй степени, 21 таксона третьей степени и 63 таксонов четвертой степени.

.1373. Разным группам таксонов присвоены названия и буквенные идентификаторы типа. Как обращаться с именами, читателю интуитивно понятно. К примеру, ему будет ясно, что такое «однородный файл переменной неограниченной длины». Именование осуществляется комбинированием названий. Точно так же надо поступать и с идентификаторами, например, «структура типа ВРТЕ» – это то же самое, что «однородный (Е) файл (В) переменной (Р) неограниченной (Т) длины».

.1374. Обратите внимание на то, что в этой классификации, как и во всех других, распределение секаторов по степеням таксонов (как и сам выбор секаторов) был в значительной степени произвольным и во многом зависел от автора классификации.

.1375.



Фиг.1. Образование таксонов третьей степени

3. Классификация ссылок

1980.03

.1376. Одни из самых главных соглашений по языку между автором текста и читателем – это соглашения о том, какие комбинации знаков в тексте каким соответствуют объектам. Структура текста, о которой между автором и читателем заключено такое соглашение, называется ссылкой, а сам объект – ее денотатом.

.1377. В этой главе мы рассмотрим классификацию ссылок.

.1378. Таксоны первой степени образуем при помощи секатора ТА (множество ссылок, которые являются отрезками):

.1379. а) простые ссылки (отрезки);

.1380. б) сложные ссылки (структуры, не являющиеся отрезками).

.1381. Таксоны второй степени образуем при помощи секатора ТВ (множество ссылок, денотатом которых являются структуры этого же текста):

.1382. а) внутренние ссылки (на структуры рассматриваемого текста);

.1383. б) внешние ссылки (на объекты вне текста).

.1384. Таксоны третьей степени образуем при помощи секатора ТС (множество символов, построенных по позиционному алгоритму). Образование символов по позиционному принципу заключается в следующем: предполагается, что между автором и читателем уже существует соглашение о том, каким образом денотаты расположены в линейный ряд. Избирается некоторое подмножество букв алфавита, о которых также предполагается, что они расположены в линейный ряд. Первому денотату присваивается символ, состоящий из знака первой буквы этого подмножества алфавита, второму – из знака второй буквы и т.д. Когда все буквы избранного подмножества алфавита исчерпаны, строим двухзначные (потом трехзначные и т.д.) символы, комбинируя первую (потом две первые и т.д.) букву избранного подмножества алфавита со всеми остальными буквами. Символ, образованный по этому принципу, называется номером. Как видите, в этом понимании номер не имеет никакого отношения к числам. Наоборот, позиционная нумерация чисел – один из примеров применения этого принципа. Позиционный принцип образования символов позволяет (при достаточной длине символа) заключать соглашения по обозначению произвольно длинного ряда денотатов, причем, заранее не перечисляя все возможные символы.

.1385. Итак, ссылки делятся на:

.1386. а) номера или номерные ссылки (построенные по позиционному принципу);

.1387. б) имена или именные ссылки (построенные по какому-нибудь другому принципу).

.1388. Денотаты могут сами включать структуру текста, которая является символом из той же Морфемы, что и ссылка на денотат. Такая структура называется меткой.

.1389. Таксоны четвертой степени в классификации ссылок образуем секатором ТМ (множество ссылок на денотаты с меткой):

.1390. а) идентификаторы (ссылки на денотаты с меткой);

.1391. б) адреса (ссылки на денотаты без метки).

.1392. Для каждой ссылки можно образовать два множества:

.1393. а) множество МО ссылок, символы в которых принадлежат к той же морфеме;

.1394. б) множество МЕ ссылок на тот же денотат.

.1395. Таксоны пятой степени в классификации ссылок образуем при помощи секатора ТО (множество ссылок, у которых дополнение МЕ в МО пусто):

.1396. а) уникальные ссылки (дополнение пусто, все символы данной морфемы везде обозначают один и тот же денотат);

.1397. б) омонимные ссылки (имеются символы этой морфемы, обозначающие другой денотат, в разных местах одинаковыми символами обозначаются разные денотаты).

.1398. Таксоны шестой степени образуем при помощи секатора ТЕ (множество ссылок, у которых дополнение МО в МЕ пусто):

.1399. а) единообразные ссылки (дополнение пусто, данный денотат везде обозначается только символами этой морфемы);

.1400. б) синонимные ссылки (денотат обозначается разными символами).

.1401. Такова та классификация ссылок, которую я буду в дальнейшем использовать. В ней 2 таксона первой, 4 таксона второй, 8 таксонов третьей, 16 таксонов четвертой, 32 таксона пятой и 64 таксона шестой степени.

4. Разновидности структур

1980.03

.1402. Внутренние ссылки используются для заключения динамических соглашений о порядке чтения текста (предварительным соглашением по этому вопросу можно считать соглашение «читать подряд, пока не встретится указание перескочить»), или для связывания записей файлов и облаков.

.1403. Способы установления связей между различными структурами чрезвычайно разнообразны и очень трудно дать какую-то их классификацию. В этой главе я вкратце рассмотрю те способы, с которыми нам придется в основном иметь дело в Эуклидосе.

.1404. Однородные или гомогенные массивы фиксированной или переменной длины называются векторами (значит, векторы характеризуются тем, что их записи имеют одинаковую длину; число и длина записей известна к началу чтения).

.1405. Если одна структура включает ссылки на все записи другой структуры, то первая называется каталогом, а вторая – библиотекой. Если одна структура A включает ссылку на первую запись другой структуры B, а каждая запись структуры B (кроме последней) включает ссылку на следующую, то первая структура (A) называется указателем (списка), а вторая (B) – списком.

.1406. Если указатель списка B входит в запись другого списка C, то список B называется подсписком списка C. Если указатель списка C входит в запись списка E, то объединение списка C со всеми его подсписками (и с подсписками тех и т.д.) называется веткой списка E. Объединение списка E со всеми его ветками называется деревом, а сам список E – стволом (схему см. Фиг.2 {.1416}).

.1407. Все эти названия относительны: дерево может быть веткой другого дерева и т.д.; они имеют смысл лишь относительно какого-нибудь списка, избранного стволом.

.1408. Дерево называется простым, если во всех входящих в него списках нет записи, которая включала бы более, чем один указатель подсписка.

.1409. Подсписки ствола я буду называть подсписками глубины 1, их подсписки – подсписками глубины 2 и т.д. Я буду говорить, что дерево имеет глубину p, если в нем имеются подсписки глубины не больше, чем p.

.1410. Записями дерева являются подсписки. Дерево – вид облака.

.1411. Куст – это структура, которую можно рассматривать как деформированное дерево: ствол включает только одну запись, которая имеет подсписки только глубины 1, но зато этих веток много (схему см. Фиг.3 {.1417}).

.1412. Бахрома – еще одна структура, часто употребляемая в Эуклидосе (см. Фиг.4 {.1418}). Это список, записи которого содержат только ссылки на другие структуры.

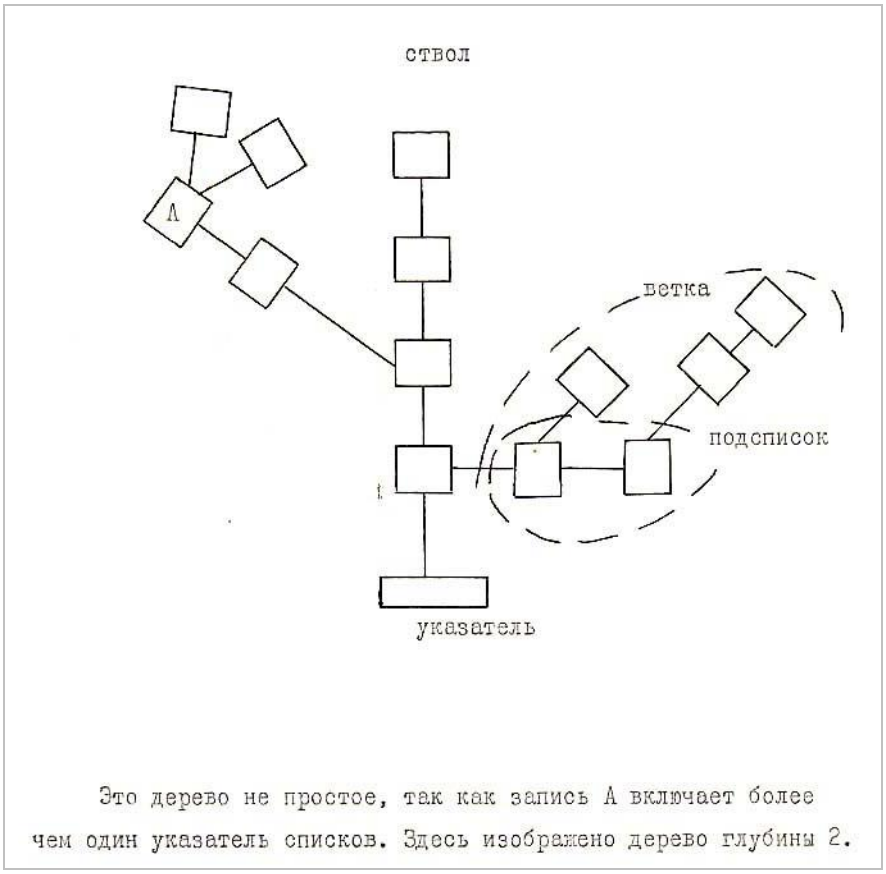
.1413. Эти не очень точно определенные понятия (списки, деревья и т.д.) хорошо знакомы программистам, и соответствующие структуры часто встречаются во внутримашинных текстах. Вне ЭВМ широко используются только структуры типа «библиотека-каталог», но в принципе нет никаких трудностей в том, чтобы представить себе реализацию и других структур текстов вне машины, например, вообразите себе книгу, в конце (или в начале) каждой страницы которой указан номер той страницы, которую надо читать следующей (к примеру, в конце страницы 7 стоит ссылка: «продолжение на странице 121», а в конце с.121 стоит ссылка: «продолжение на с.33» и т.д.).

.1414. Такая книга была бы организована по принципу списка. Разумеется, реально в такой организации книги нет необходимости, потому что место, «память» книги распределена раз и навсегда, и не меняется динамически. Реальным примером списковой организации текстов вне машины могут послужить романы, печатаемые в газетах или в журналах по продолжениям.

.1415. Итак, рассмотренные в предыдущих главах классификации структур и ссылок в текстах ни в коем случае не относятся к какой-нибудь одной разновидности текстов (например, к

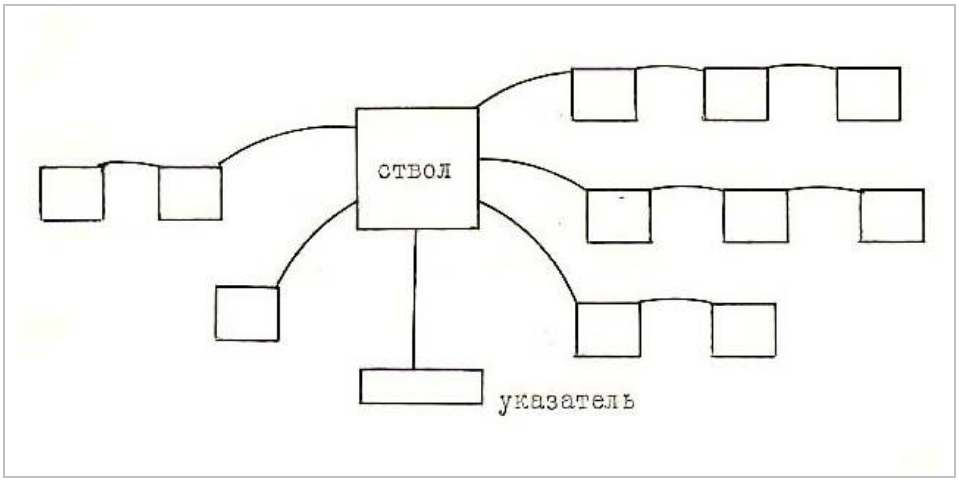
текстам во внутримашинных интерфейсах). Всё сказанное выше относится в равной мере к любому тексту на каком бы носителе он ни был записан.

.1416.



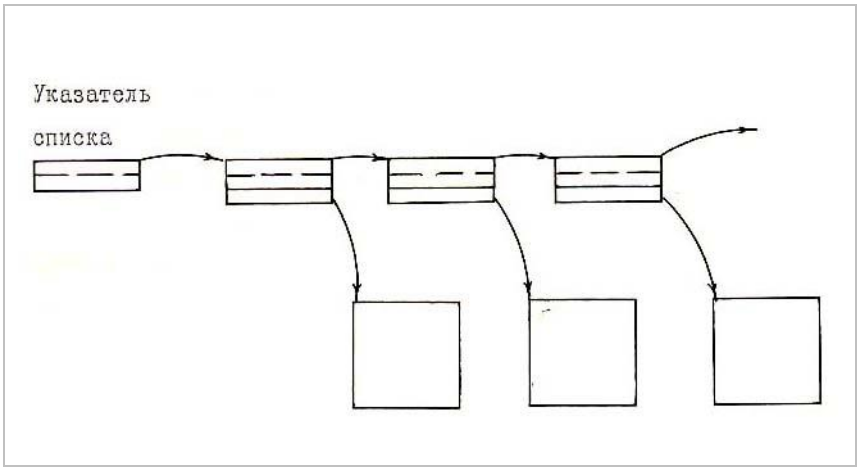
Фиг.2. Схема дерева

.1417.



Фиг.3. Схема куста

.1418.



Фиг.4. Схема бахромы

5. Алфавит Эуклидола

1980.03

.1419. Теперь, после столь длительного раздумья над тем, что такое вообще алгоритмический язык и тексты, мы пришли, наконец, к тому месту, с которого обычно начинают описание алгоритмического языка – к алфавиту или к началу теории текстов.

.1420. Итак, задача: построить Здание теории {.516}, то есть, некоторую совокупность множеств над некоторым исходным множеством, которое называется пространством этой теории и состоит из точек этой теории. В нашей теории точками являются знаки текста, а пространством – некоторое множество знаков – текст на Эуклидоле. Этот текст существует в Главном интерфейсе (написан на бумаге). Иногда я буду рассматривать также преобразования этого (канонического) текста в других интерфейсах, но предмет нашей теории – канонический текст и множества, построенные над ним.

.1421. В первую очередь рассмотрим группировку пространства (знаков текста), состоящую из 77 таксонов. Любой знак принадлежит одному из этих таксонов. Описание алгоритмов, по которым мы можем определить принадлежность знака к тому или иному таксону, опустим, хотя нет никаких сомнений, что эти алгоритмы, по которым мы можем отличить, например, знак «а» от знака «е», совершенно реальны. Итак, имеем 77 непересекающихся множеств первого уровня, но алгоритмы их построения в нашей теории не описаны.

.1422. Множество (уже второго уровня), состоящее из 76 таксонов, называется алфавитом Эуклидола. 77-ой таксон содержит все те знаки, которые не принадлежат ни одной из букв алфавита Эуклидола.

.1423. Множество из всех 77 таксонов я буду называть алфавитом текста. Таким образом, нужно различать алфавит Эуклидола и алфавит текста.

.1424. Рассмотрим некоторую группировку алфавита текста (группировка букв). В этой группировке 10 таксонов. Буквы – это множество первого уровня; группировка их – уже множество второго уровня. Объединение всех букв одного таксона этой группировки называется типом знака. Тип знака – опять множество первого уровня, то есть – это множество знаков. Ниже приводится перечень букв по типам, в которые они входят.

.1425. 1) Тип BA объединяет 48 букв: ABCEHKMOPTX DFGIJLNQRSUVWYZ ЮБЦДФГИЙЛПЯУЖЪЬЫЗШЭЩЧ№. (Знаки Ъ и & принадлежат одной букве; знаки № (*русский знак номера, имевшийся на пишущих машинках – ред.*) и # также принадлежат одной букве).

.1426. 2) Тип BM объединяет 10 букв: 0123456789. (Знаки этих букв называются также цифрами).

.1427. 3) Тип BP объединяет 10 букв:

( ) ” ! ? + – / = *

,которые называются соответственно: открывающая скобка, закрывающая скобка, кавычки, восклицательный знак, вопросительный знак, плюс, минус, дробь, знак равенства и звездочка.

.1428. 4) Тип ВК объединяет 3 буквы:

, :

– запятую, пробел и двоеточие.

.1429. 5) Тип ВЕ включает одну букву – дефис (имеет начертание: «-» или «'»).

.1430. 6) Тип ВТ включает одну букву – точку.

.1431. 7) Тип ВС включает одну букву – точку с запятой, которую я впредь ради краткости называю семиколоном.

.1432. 8) Тип ВВ включает одну букву – параграф, который может иметь три начертания: § (параграф), $ (доллар) или ¤ («рубль», «солнышко»).

.1433. 9) Тип ВО включает одну букву – процент %.

.1434. 10) Тип ВН включает одну букву, не принадлежащую алфавиту Эуклидола, то есть ту, которой принадлежат все «чужие» знаки.

.1435. Тип ВА иногда рассматривается как объединение четырех подтипов – подтип общих букв, подтип латинских букв, подтип русских букв и подтип цифр.

.1436. Итак, я определил состав алфавита Эуклидола. Динамические соглашения по алфавиту в Эуклидоле запрещены; разница между заглавными и строчными буквами игнорируется, то есть, знак «А» и знак «а» считаются одной и той же буквой; употребление того или иного знака не несет в Эуклидоле никакой смысловой нагрузки, и для него совершенно безразлично, написали ли Вы «Язык» или «яЗЫК» или «ЯЗЫК» или еще как-нибудь.

.1437. К сожалению, некоторые трудности остаются и при таком алфавите. Конструкция некоторых печатающих устройств настолько неудачна, что невозможно отличить напечатанные ими букву О и цифру 0; латинскую букву I и цифру 1, дефис и минус; другие печатающие устройства, рассчитанные на русский алфавит, не имеют законной русской буквы Ъ. В общем, вопрос алфавита в реальной жизни находится в обычном хаотичном и нелогичном состоянии.

.1438. Поэтому не остается ничего другого, как в текстах Эуклидола, напечатанных на пишущей машинке и рассчитанных на человека, положиться на то, что читатель сможет сам разобраться, где дефис, а где минус, где единица и где буква I, а в текстах Эуклидола, рассчитанных на ЭВМ или выданных ею, считать, что не имеющиеся в коде ДКОИ буквы Эуклидола имеют другое начертание по сравнению с пишущей машинкой, а именно: дефис как апостроф (знак '), буква Ъ как амперсанд (знак &), параграф как знак $ (или «рубль» кода ДКОИ), а номер как знак #.

.1439. Итак, одной и той же букве принадлежат:

.1440. а) знак «-» и знак «'» (это дефис);

.1441. б) знак «Ъ» и знак «&»;

.1442. в) знак «§», знак «$» и знак «¤» (это параграф).

.1443. Хотя в алфавит Эуклидола я включил и латинские и русские буквы, но при прочих равных условиях я, в первую очередь, пользуюсь буквами, имеющимися в обоих алфавитах. При изложении не с фундаментальной точностью (не на Эуклидоле) я также стараюсь обойтись алфавитом Эуклидола, а при выборе символов напирать на буквы АВСЕНКМОРТХ и асекорх.

6. Подпредложения Эуклидола

1980.03

.1444. Текст, написанный на Эуклидоле, всегда рассматривается как линейная последовательность (порядок) знаков. Если физически эта последовательность разрывается на строки, расположенные сверху вниз, то такой разрыв эквивалентен пробелу, вставленному в этом месте в последовательность.

.1445. Фрагментом на Эуклидоле называется отрезок (непрерывная непустая последовательность знаков) от знака § включительно до знака % включительно (см. Фиг.5 {.1484}). Таким образом, в текст входит некоторое количество фрагментов и некоторое множество знаков, не принадлежащих ни одному фрагменту. Это множество называется полями текста на Эуклидоле. Знаки § и % разбивают текст на фрагменты и поля.

.1446. Отрезок, входящий во фрагмент (непустое множество подряд идущих знаков), находящийся между любыми двумя знаками типов ВТ, ВС, ВВ или ВО, называется подпредложением. Таким образом, во фрагмент входит некоторое количество подпредложений и знаки

типов ВТ, ВС, ВВ, ВО. Эти знаки (точка, семиколон, параграф, процент) разбивают фрагмент на подпредложения, но сами подпредложению не принадлежат.

.1447. Подпредложение, непосредственно перед первым знаком которого стоит семиколон, называется комментарием. Все подпредложения делятся на высказывания и комментарии. Комментарии начинаются после семиколона и, как правило, заканчиваются точкой.

.1448. Итак, над исходным пространством мы построили:

.1449. а) множество фрагментов (сам фрагмент – множество уровня 1; множество их – уровня 2);

.1450. б) множество, называемое «поля» (уровня 1);

.1451. в) множество знаков ВТ, ВС, ВВ, ВО;

.1452. г) множество подпредложений (уровня 2; сами подпредложения – уровня 1);

.1453. д) группировку (множество уровня 3) множества подпредложений, состоящую из двух элементов: множества высказываний и множества комментариев.

.1454. Разумеется, что это множества абстрактные; они заданы лишь алгоритмом, пока у Вас нет конкретного материала, конкретного текста. Получив конкретный текст, Вы по этим алгоритмам могли бы построить уже конкретные множества.

.1455. Алгоритмы, которыми эти множества задаются, конечно, выше не были явно описаны. Я говорил: «отрезок между знаками ВТ называется подпредложением...» и т.п. Но я мог бы то же самое сказать и так: «просматривайте последовательность знаков, пока не наткнетесь на знак ВТ, все дальнейшие знаки помещайте в подпредложение номер один, пока не наткнетесь на новый знак ВТ...». Это уже намного больше похоже на описание алгоритма, не правда ли? Любой программист легко представит себе, как можно описать это еще более точно при помощи команд типа машинных.

.1456. Любое определение понятия – всегда описание алгоритма, и если оно не похоже на таковое, то только потому, что люди не всегда отдают себе в этом отчет.

.1457. Поля и фрагменты, высказывания и комментарии разбиваются на более мелкие структуры по-разному (то есть, они обрабатываются при чтении по разным алгоритмам). Алгоритмы теории текстов Эуклидола предписывают в полях текста реагировать только на один знак: § – начало фрагмента. Все остальные знаки в полях при чтении текста по алгоритмам Эуклидола бесследно пропускаются (конечно, они могут не пропускаться, если Вы читаете не по правилам-алгоритмам Эуклидола, а по каким-то другим правилам, например, по алгоритмам русского языка).

.1458. Таким образом, можно, например, чередовать фрагменты Эуклидола с русскими текстами (если только в них нет знака §), и при этом ничуть не грешить против жестких законов Эуклидола. Именно так я и буду поступать. В этом предложении знак §, приведенный «просто так», Вы видите в последний раз; всюду впредь его появление означает начало фрагмента на Эуклидоле (*это было так в Третьей Медиотеке – ред. – но в Ведде это не всегда так, ибо в теках этот знак используется самостоятельно*).

.1459. Внутри фрагмента читатель, просматривающий текст по правилам Эуклидола, уже должен реагировать на все знаки, по которым выделяются подпредложения (типы ВТ, ВС, ВВ, ВО) и на знак типа ВН, который внутри фрагмента (в отличие от полей) должен восприниматься как нарушение правил Эуклидола, то есть – как ошибка в тексте.

.1460. Внутри комментариев он должен реагировать только на эти знаки, пропуская все остальные. Внутри высказываний же он ведет полную дешифровку текста.

.1461. Итак, можно считать, что правила Эуклидола предписывают три типа просмотра текста:

.1462. а) в полях – игнорировать всё, кроме начала фрагмента;

.1463. б) в комментариях – игнорировать всё, кроме конца подпредложения и недопустимого знака;

.1464. в) в высказываниях – обрабатывать всё.

.1465. В дальнейшем я буду рассматривать только обработку высказываний.

7. Поток символов

1980.03

.1466. Теперь рассмотрим множество, полученное как объединение всех высказываний текста. Это множество называется чистым текстом Эуклидола. Это текст, из которого выброшены все поля, комментарии и знаки ВТ, ВС, ВВ, ВО.

.1467. Разделителем называется отрезок (непустое множество подряд идущих знаков), состоящий только из знаков типа ВК (пробелы, запятые и двоеточия в любом количестве и сочетании). Напомню, что сам разрыв строки эквивалентен пробелу.

.1468. При помощи разделителей, дефиса и знаков типов ВР и ВН в чистом тексте Эуклидола выделяются множества знаков, называемые символами.

.1469. Главным средством для выделения символов служат три вспомогательные буквы – пробел, запятая и двоеточие. Везде, где есть хотя бы один из этих знаков, может быть любая их последовательность. Таким образом, символы отделяются группами пробелов, запятых, двоеточий или их смеси. Употребление того или другого знака зависит только от желания автора. Разрешение присутствия запятых и двоеточий я предусмотрел для того, чтобы иметь возможность сделать текст, написанный на Эуклидоле, хоть немножко более читабельным для человека.

.1470. Дефис (знак «-» или апостроф кодов ДКОИ или EBCDIC) употребляется для аннулирования группы пробелов, запятых или двоеточий (разделителя), следующего непосредственно за ним. Группа знаков, предшествующая дефису, и группа знаков, следующая за группой разделителя, считается одним символом. Это можно использовать для переноса символов, не дожидаясь конца карты. Если же за дефисом не следует разделитель (пробел, запятая или двоеточие), то дефис считается рядовым знаком. Это можно использовать для образования единых для машины, но составных для человека слов (например: «Язык-Эуклидол»). Если требуется такой перенос этого слова, что первая его часть заканчивается на дефисе, то следует ставить подряд два дефиса.

.1471. Итак, если разделитель следует непосредственно за знаком типа ВЕ (дефисом), то дефис вместе с разделителем опускается, и знаки, предшествующие дефису и следующие за разделителем, считаются принадлежащими одному символу или «соседними».

.1472. Символом типа ХА (буквенным символом или именем) в Эуклидоле называется любая последовательность (непустое множество) подряд идущих или соседних знаков типов ВА, ВМ и ВЕ, если такая последовательность не состоит из одних только цифр.

.1473. Символом типа ХМ (числовым символом или числом) называется любая последовательность (непустое множество) подряд идущих или соседних знаков типа ВМ (цифр).

.1474. Символом типа ХР (специальным символом) называется один знак типа ВР.

.1475. Итак, знаки типа ВР считаются отдельными символами и не могут быть использованы для образования других символов (последовательностей знаков). Они от предшествующих и последующих символов могут и не отделяться разделителями (но могут и отделяться – как автор пожелает). При отсутствии разделителей символы ХР сами служат ограничителями для соседних символов. В этом смысле знаки типа ВН (недопустимые знаки) похожи на знаки ВР – они тоже служат ограничителями для соседних символов, если попадаются в тексте.

.1476. Символом типа ХТ (конечным символом) называется конец предложения (пустое множество знаков, находящихся между последним знаком подпредложения и знаком типа ВТ, ВС, ВВ, ВО).

.1477. Множество всех символов (типов ХА, ХМ, ХР и ХТ), входящих в одно непустое высказывание, называется предложением Эуклидола, а объединение всех предложений (множество всех символов текста) называется потоком Эуклидола. В предложение входят один или несколько предикатов. Правила расчленения предложения на предикаты будут рассмотрены ниже.

.1478. Символы – это множество уровня 1, поток, предложения и предикаты – множества уровня 2.

.1479. Эуклидос может накладывать на язык некоторые ограничения. Например, в нем длина символа не должна быть больше определенного числа знаков, числа в десятичной записи должны иметь не более 9 знаков (большие числа не могут быть записаны в регистр машины) и т.д. Сам же язык Эуклидол не нуждается в таких ограничениях. Такие правила я в описаниях

помечаю словами «ограничение Эуклидоса»; их соблюдение необязательно в текстах Эуклидола, предназначенных для людей.

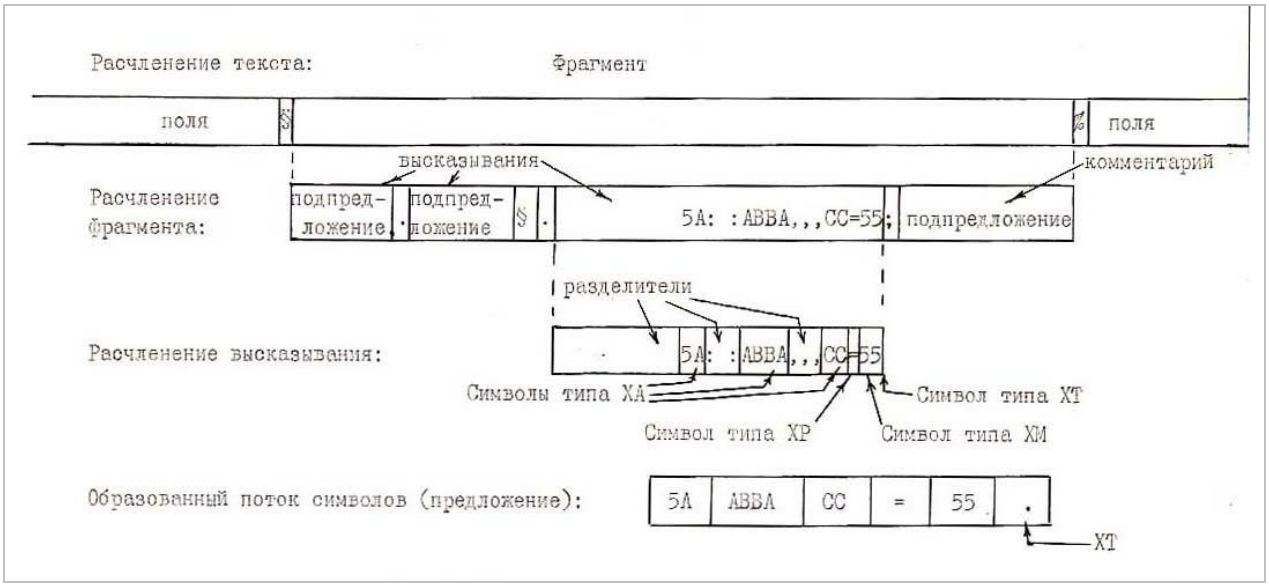
.1480. Максимальная длина символа в Эуклидосе может быть легко изменена (заменой одной единственной перфокарты). В этих описаниях я буду считать, что максимальная длина символа установлена в 16 знаков, но это число в значительной степени условно. Окончательное значение этого числа см. в автоскриптах Эуклидоса.

.1481. Основная масса символов образовываются путем комбинирования знаков 58 букв. При максимальной длине символа 16 знаков всего могут быть образованы $10 + 58 + 58^2 + ... + 58^{16}$ или примерно 10^{29} морфем.

.1482. В дальнейшем я буду рассматривать только поток символов, не возвращаясь больше к вопросам их выделения. Поток, символы, предложения, предикаты станут главными объектами дальнейших разговоров об Эуклидоле. Можно даже считать, что мы начинаем рассмотрение новой теории, точками которой являются символы, пространством – поток, и над пространством каким-то алгоритмом задана группировка символов на четыре типа: XA, XM, XP и XT (последние замыкают предложение; предложение обязательно содержит хотя бы один символ типа XA, XM или XP, два XT не могут идти подряд).

.1483. Поэтому еще раз зафиксируем принципы определения символов в текстах Эуклидола, на этот раз с точки зрения системы Эуклидос.

.1484.



Фиг.5. Расчленение потока знаков

8. Выделение символов Эуклидосом

1980.03

.1485. Теперь посмотрим, что означает это всё для Эуклидоса.

.1486. Транслятор №1 (перфораторщица с перфоратором) переводит канонический текст на язык перфокарт {.1137}. Реально, видимо, этот транслятор опустит все поля текста. Трудно предсказать, как он будет реагировать на знаки типа BH, если их нет на клавиатуре перфоратора. В остальном знаки канонического текста перекодируются один к одному в знаки текста на перфокартах.

.1487. Транслятор №2 (устройство ввода карт) в исправном состоянии проявит еще меньше инициативы, и в результате можно надеяться, что отрезок из восьми битов будет в интерфейсе №3 (интерфейсе системы) однозначно соответствовать одному знаку канонического текста.

.1488. Этот текст в неизмененном виде будет передан в интерфейс №4 (интерфейс Эуклидоса). Точно в таком же виде карты текста появятся в интерфейсе №4, если они от

канонического текста шли не путем бумага–карты–память, а каким-нибудь другим путем, например, бумага–дисплей–память или бумага–карты–память–диск–память.

.1489. Теперь настало время, как было обещано {.1140} продолжить каскад трансляторов и интерпретаторов (см. Фиг.6 {.1505}). Задача Супервизора заключается в том, что он сливает в единый поток поступающие от разных источников и из эуклидотеки карты текста. В интерфейсе №5 уже имеется только один общий входной поток. В остальном карты проходят через Супервизор неизмененными.

.1490. Собственно разбор текста Эуклидола начинается только после интерфейса №5, когда начинается выделение символов из текста, преобразование потока знаков в поток символов.

.1491. Выделение символов осуществляет подпрограмма BATCH. Через интерфейс №5 она обращается к Супервизору и получает от него очередную карту. Размещение информации на картах значения не имеет, так как подпрограмма BATH рассматривает входной поток как бесконечный ряд знаков.

.1492. В интерфейс №6 подпрограмма BATCH выдает символ в виде буфера динамически выделяемой оперативной памяти (буфера XBOP), где в поле XX находится сам символ (знаки его); в байте XCM биты XM и XP сигнализируют о символах соответствующих типов, а их отсутствие – о типе XA (символы типа XT в буфер не помещаются, о них BATCH сигнализирует признаком результата). В буфере содержится и другая информация о символе. За одно обращение BATCH выделяет один символ.

.1493. Оформленный таким образом буфер в дальнейшем попадает к рабочим модулям, но непосредственно к BATCH рабочие модули не обращаются. Входной поток они читают через интерфейс №7 при помощи подпрограммы BAOBAB. Эта подпрограмма из буферов, полученных ею от BATCH, строит простое дерево предиката с указателем списка в BOP. Ствол дерева образовывает буфера, содержащие операнды предиката; от операндов могут отходить ветки подоперандов и т.д. Буфер с самим оператором находится в отдельном списке BOPO (подробнее об этом в следующих главах {.1525}, {.1543}). Рабочие модули, таким образом, читают входной поток текста Эуклидола через интерфейс N7 целыми синтаксически проверенными предикатами в виде деревьев. Но листья этих деревьев подготавливает подпрограмма BATCH.

.1494. Итак, те сложные и скучные правила, определяющие, что такое символ, о которых я нудно писал выше – это на самом деле не что иное, как описание алгоритма, который с одной стороны воплощен в подпрограмме BATCH, и который, с другой стороны, должен выработать у себя любой другой читатель текстов на Эуклидоле. Об этом алгоритме я намеренно говорил исключительно в терминах теории множеств. Такими средствами я стараюсь привести читателя к убеждению, что разговоры о множествах и об алгоритмах – это одно и то же.

.1495. Но вернемся к подпрограмме BATCH и взглянем еще раз с ее точки зрения на поток знаков текста. Итак, первые соглашения Эуклидола с точки зрения системы Эуклидос:

.1496. 1) Алфавит языка состоит из 76 букв:

а) 11 общих букв ABCEHKMOPTX
б) 15 латинских букв DFGIJLNQRSUVWYZ
в) 21 русская буква ЮБЦДФГИЙЛПЯУЖЪЬЫЗШЭЩЧ
г) 1 буква номера #
д) 10 цифр 0123456789
е) 10 специальных букв ()!?»+–/=*
ж) 8 служебных букв , : . ; ¤ % ' (первым стоит пробел)

.1497. 2) Остальные 180 знаков кода EBCDIC (ДКОИ) внутри фрагмента воспринимаются как ошибка в исходном тексте, а в полях игнорируются.

.1498. 3) Четыре служебных знака («параграф», «процент», «точка» и «семиколон») служат для выделения в потоке знаков фрагментов, высказываний и комментариев, но одновременно и разграничивают символы.

.1499. 4) Четыре служебных знака: «пробел», «запятая», «двоеточие» и «апостроф» (играющий роль дефиса) служат для выделения в этом потоке символов. Последовательность из любого числа рядом стоящих пробелов, запятых или двоеточий в любом сочетании считается разделителем. Апостроф в исходном тексте игнорируется вместе с разделителем, если тот следует сразу за апострофом, и сохраняется в противном случае.

.1500. 5) Десять специальных знаков считаются отдельными символами и служат признаком конца предыдущего символа при отсутствии между ними разделителя.

.1501. 6) Если за просмотр 17 знаков символа в исходном потоке не встретился ни разделитель, ни специальный знак, то 17-й знак считается первым знаком нового символа.

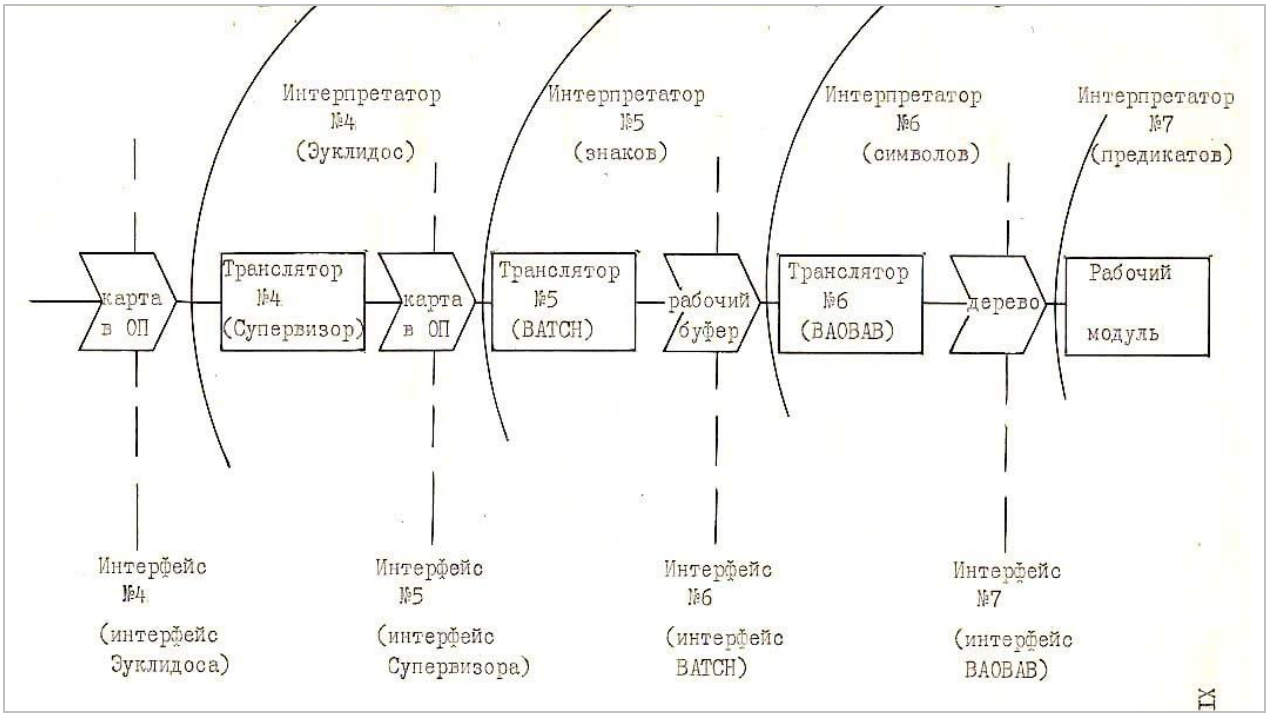
.1502. 7) Знак, не принадлежащий алфавиту Эуклидола, игнорируется, но разграничивает символы.

.1503. 8) Таким образом, разграничение символа может произойти:

а) по разделителю;
б) по специальному символу;
в) по концу высказывания;
г) по истечении 16 байтов;
д) по знаку не из алфавита Эуклидола.

.1504. В дальнейшем исходный текст Эуклидола рассматривается как последовательность не знаков, а символов, у которых длина от 1 до 16 знаков.

.1505.



Фиг.6. Иерархия внутри Эуклидоса

9. Предикаты Эуклидола

1980.04
(через 1 месяц)

.1506. Итак, в интерфейсе BATCH (и отныне у нас) уже нет никаких знаков, комментариев и т.д. Есть только поток символов, которые могут быть четырех типов:

а) XA – имя;
б) XM – число;
в) XP – специальный символ;
г) XT – конец предложения.

.1507. Символами XT поток разделен на предложения так, что XT всегда последний символ предложения. В дальнейшем расчленении потока символов особую роль играет специальные символы «(» и «)».

.1508. Скобки разбивают символы на РЯДЫ символов разной глубины (см. Фиг.7 {.1524}). От начала предложения идет первый ряд. Если появляется символ «(», то после него начинают идти символы второго ряда, если еще раз появляется символ «(», то после него идут уже символы третьего ряда, и т.д.: с каждой новой открывающей скобкой глубина ряда увеличивается, а

закрывающая скобка, наоборот, снижает глубину ряда на единицу, и когда число закрывающих скобок становится равным числу открывающих скобок, опять идет первый ряд символов.

.1509. Каким бы рядом ни закончилось предыдущее предложение, новое всегда начинается с первого ряда. Сами скобки, как и символ XT, рядам не принадлежат. (Закрывающая скобка принадлежит первому ряду, если она появляется до открывающей скобки; другим рядам принадлежать она не может).

.1510. Символы первого ряда называются членами предложения. Они подразделяются на два подмножества:

а) операторы;

б) операнды.

.1511. Считается, что каждый операнд относится к одному и только одному оператору. Считается, что все символы второго ряда являются подоперандами того операнда, непосредственно за которым они следуют. Аналогично все символы третьего ряда считаются подоперандами того подоперанда второго ряда, непосредственно за которым они следуют, и т.д.

.1512. Правила Эуклидола требуют:

.1513. а) чтобы последним символом в любом предложении перед XT был бы символ первого ряда или закрывающая скобка, выводящая поток в первый ряд (соблюден баланс скобок);

.1514. б) чтобы непосредственно перед любой открывающей скобкой находился символ какого-нибудь ряда (открывающая скобка не может идти сразу после начала предложения или после другой открывающей или закрывающей скобки);

.1515. г) чтобы открывающая скобка не следовала сразу за оператором.

.1516. Множество символов, составленное из одного оператора и всех его операндов вместе со всеми их подоперандами (и подоперандами тех и т.д.) называется предикатом. Таким образом, предложение включает один или несколько предикатов. Каждый предикат имеет один главный член – оператор, который определяет всю суть и содержание предиката. Оператор и его операнды в первом ряду следуют подряд (то есть в потоке символов между ними могут находиться их подоперанды, но не члены другого предиката).

.1517. Обычно в алгоритмических языках применяются довольно разнообразные конструкции, которые призваны облегчить чтение и понимание языка человеком, подстраиваясь под привычные ему традиционные стереотипы. Но такое разнообразие значительно затрудняет трансляцию. В Эуклидоле я не следовал этим образцам; здесь всё построено на однообразных, стереотипных конструкциях: оператор плюс ряд операндов с заключенными в скобках подоперандами, их подоперандами и так до любой глубины. Я думаю, что это единообразие в конечном счете поможет даже человеку читать текст на Эуклидоле, не говоря уже об Эуклидосе.

.1518. Обычно в алгоритмических языках используются три основные типа конструкций:

.1519. а) инфиксная, где оператор находится посередине, а операнды – с обеих сторон от него (например: A + B);

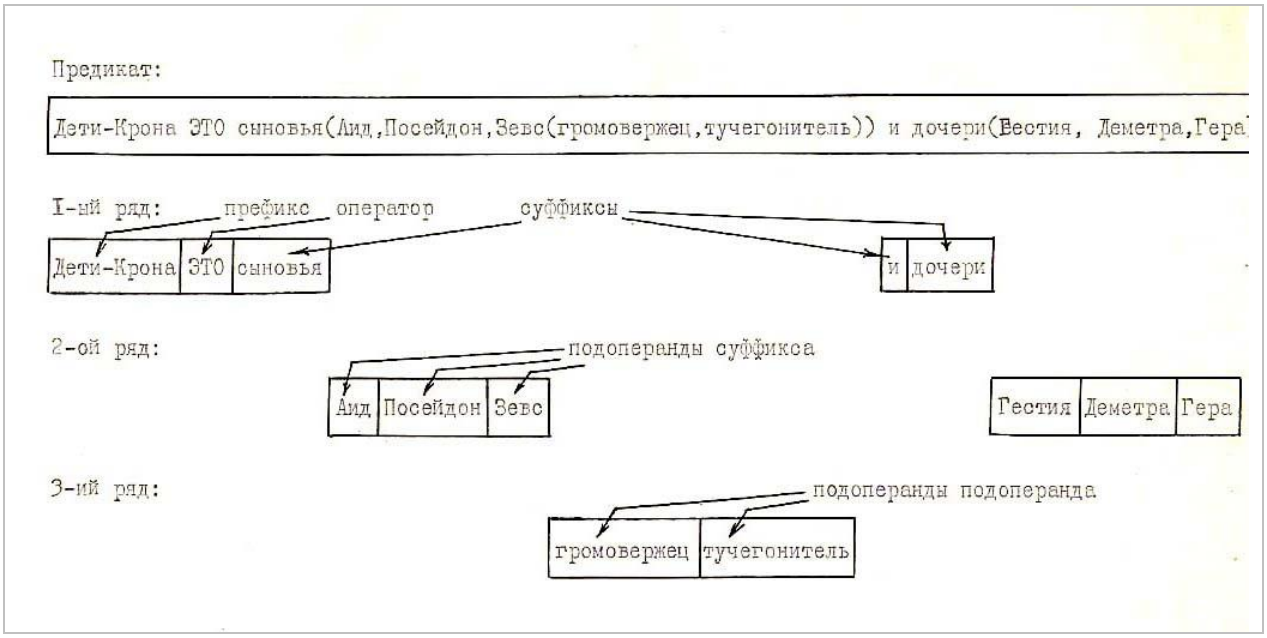
.1520. б) префиксная, где операнды предшествуют оператору (например: A, B СЛОЖИТЬ);

.1521. в) суффиксная, где операнды следуют за оператором (например: СЛОЖИТЬ A, B).

.1522. Обычно префиксная и суффиксная формы допускают любое число операндов, а инфиксная – только два.

.1523. В Эуклидоле считается, что любое высказывание построено в инфиксной форме с любым числом операндов: сначала идет некоторое число операндов, называемых ПРЕФИКСАМИ, потом оператор, и после него операнды, называемые суффиксами. Предикаты, в которых префиксы или суффиксы отсутствуют, считаются частными вариантами общего случая.

.1524.



Фиг.7. Расчленение предиката

10. Типы операторов

1980.04

.1525. Выделение предикатов из предложения осуществляет подпрограмма ВАОВАВ. В интерфейсе №7 она выдает готовые предикаты в виде структур типа «дерево»: ствол составляет список операндов, от них отходят подсписки подоперандов и т.д. (см. Фиг.8 {.1543}). В ней осуществлен алгоритм, определяющий множество предикатов.

.1526. В выделении предикатов людьми, конечно, большую роль играют размещение предиката в отдельной строчке и подобные способы, не имеющие никакого отношения к правилам Эуклидола, но с формальной точки зрения предикат определен тем алгоритмом, который описан ниже.

.1527. Главным ориентиром в выделении предикатов является оператор. Число разновидностей операторов и их названия известны. ВАОВАВ (и вообще любой читатель) просматривает поток символов от начала предложения или от конца предыдущего предиката, пока не наткнется на символ, обозначающий оператор и идущий в первом ряду предложения. (Чтобы облегчить человеку поиск оператора, везде в текстах Эуклидола его название я буду писать заглавными буквами).

.1528. Каждый оператор имеет определенные характеристики: у одних число префиксов или суффиксов, или тех и других, фиксировано, у других – произвольно. У одних глубина вложенности подоперандов у префиксов и/или суффиксов фиксирована, у других – не ограничена; у одних суффиксы и префиксы эквивалентны, у других имеют разный смысл. Всё это становится известным, когда обнаружен оператор (Эуклидос имеет таблицу операторов, где хранятся сведения о каждом операторе).

.1529. Имеются шесть группировок операторов по типам:

.1530. 1) По эквивалентности префиксов и суффиксов:
- а) тип ОЕ – эквивалентны;
- б) тип ОО – не эквивалентны.

.1531. 2) По количеству префиксов:
- а) тип ОРЕТ – число префиксов не лимитируется,
- б) тип ОРР – число префиксов не меньше числа ОР,
- в) тип ОРМ – число префиксов не больше числа ОР,
- г) тип ОРЕ – число префиксов в точности равно числу ОР;

.1532. 3) по количеству суффиксов:
- а) тип ОКЕТ – число суффиксов не лимитируется,

б) тип ОКР – число суффиксов не меньше числа ОК,
в) тип ОКМ – число суффиксов не больше числа ОК,
г) тип ОКЕ – число суффиксов в точности равно числу ОК;

.1533. 4) по глубине вложенности подоперандов у префиксов:
а) тип ОРОЕ – глубина вложенности не ограничена;
б) тип ОРОО – число открывающих скобок не больше числа ООР;

.1534. 5) по глубине вложенности подоперандов у суффиксов:
а) тип ОКОЕ – глубина не ограничена,
б) тип ОКОО – число открывающих скобок не больше числа ООК.

.1535. 6) по порядку следования в блоке:
а) тип ОВ – отдельный (первый и последний),
б) тип ОН – головной (первый, но не последний),
в) тип ОС – промежуточный,
г) тип ОТ – хвостовой (последний, но не первый).

.1536. Последняя, шестая группировка будет подробнее описана несколько ниже {.1560}.

.1537. Числа ОР, ОК, ООР, ООК фиксированы отдельно у каждого оператора. Точное значение этих чисел и перечень всех типов, к которым оператор принадлежит, Эуклидос печатает в своих автоскриптах.

.1538. При обнаружении оператора подпрограмма BAOBAB проверяет, не противоречит ли фактическое состояние дел с префиксами тому, какое должно быть по описанию оператора, и потом продолжает просмотр потока символов до ХТ, если число суффиксов не ограничено или пока не поступит указанное число (ОК) суффиксов.

.1539. Если перед оператором типа ОРМ или ОРЕ прочитано слишком много префиксов, выдается сообщение об ошибке и игнорируются первые операнды (наиболее далеко отстоящие от оператора). Если перед оператором типа ОРР или ОРЕ прочитано слишком мало префиксов, то весь предикат игнорируется. Предикат игнорируется также, если после оператора типа ОКР или ОКЕ прочитано слишком мало суффиксов. Прочитать слишком много суффиксов невозможно, так как подпрограмма по исчерпанию счетчика ОК прекращает чтение.

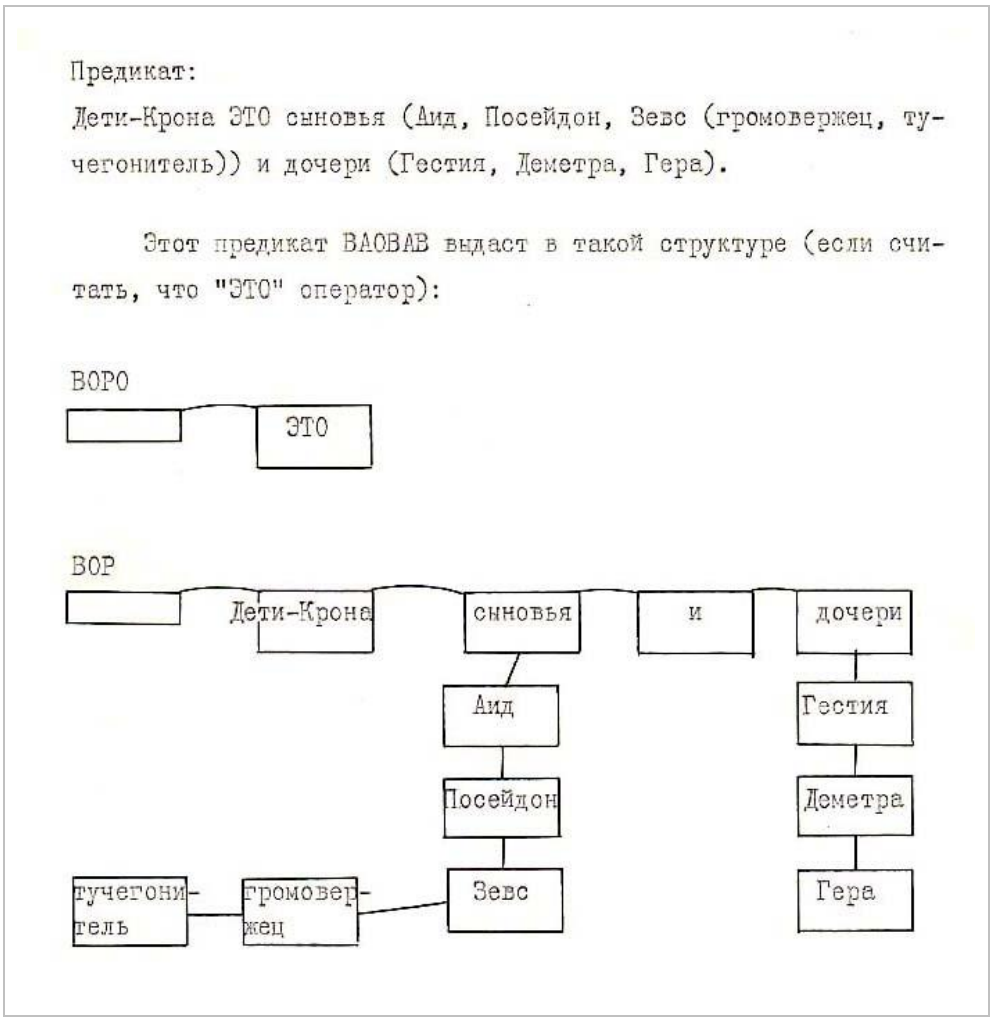
.1540. Если подоперанды появляются там, где они не должны быть (согласно описанию оператора), то подпрограмма выдает сообщение об ошибке и игнорирует подоперанды. Весь предикат целиком игнорируется подпрограммой, если он построен синтаксически неправильно: если появляются подоперанды без операнда, если встречается конец предложения внутри подоперандов, если встречается новый оператор среди операндов предыдущего до исчерпания счетчика ОК, и если оператор вообще не найден до символа ХТ. Если новый оператор появился среди операндов предыдущего, то игнорируется только старый оператор и делается попытка правильно интерпретировать новый.

.1541. Итак, путем обращения к BAOBAB модули читают входной поток в виде предикатов, в которых уже отсеяны синтаксические ошибки. В этом потоке в составе предиката различаются:

а) оператор;
б) префикс – операнд, предшествующий оператору;
в) суффикс – операнд, следующий за оператором;
г) подоперанд какого-нибудь операнда или подоперанда.

.1542. Дальнейшие действия Эуклидоса уже зависят от конкретного оператора. Как среди операндов, так и на базе подоперандов при помощи специальных символов или другими средствами могут быть организованы более сложные конструкции языка, однако все такие образования будут уже не глобальными для всего Эуклидола, а специфическими для какой-нибудь группы операторов или для отдельного оператора. Общие для всех предикатов конструкции – это только операнды (делящиеся на префиксы и суффиксы) и заключенные в скобках подоперанды.

.1543.



Фиг.8. Структура предиката

11. Названия операторов

1980.04

.1544. В высказываниях человеческих языков обычно имеются два главных члена предложения – подлежащее и сказуемое. Эуклидол и здесь упрощает принцип языка: в нем имеется только один главный член предиката – оператор. Весь остальной предикат строится и группируется вокруг операторов, поэтому их названия имеют особое значение.

.1545. Обычно в алгоритмических языках названия операторов фиксированы и не могут быть изменены пользователем. В Эуклидоле я отступил от этого принципа. Операторы в Эуклидоле имеют базисные названия, под которыми они описаны в этих медитациях и которые первоначально заданы в таблицах Эуклидоса.

.1546. При изобретении базисных имен для операторов я руководствовался следующими принципами:

.1547. а) название оператора должно состоять только из букв, имеющихся как в латинском, так и в русском алфавитах;

.1548. б) название должно быть читабельным, то есть таким, которое человек может легко произнести вслух (не таким, как это: ВМ2СХМ);

.1549. в) название оператора должно быть коротким;

.1550. г) название оператора должно быть мнемоникой, вызывающей ассоциации с его смыслом;

.1551. д) этимологически оно должно восходить к греческому или латинскому языку.

.1552. Главная трудность заключается в выполнении требования первого пункта. Это сильно затрудняет соблюдение остальных принципов. В результате базисные названия операторов не всегда являются хорошими мнемониками.

.1553. Но это не очень важно, так как все названия операторов могут быть заменены на любые другие названия, в том числе на слова любого языка, пользующегося латинским или русским алфавитом. Замену выполняет оператор OPT, единственный оператор, реализуемый аппаратом ПРЕДИКАТ. Заодно он послужит и примером оператора вообще.

.1554. Префиксы и суффиксы у оператора OPT эквивалентны, число и тех и других не ограничено, каждый операнд должен иметь в точности один подоперанд. Операнд указывает новое, а подоперанд – старое название оператора, например, оператор

OPT есть(ext),заменить(opt).

заменит названия операторов EXT и самого OPT, а оператор

ext(есть) ЗАМЕНИТЬ.

,выданный после этого, восстановит базисное название для EXT.

.1555. В Эуклидосе имеется возможность при загрузке автоматически выполнить ряд операторов, каталогизированных в эуклидотеке, в том числе можно таким способом выполнить операторы OPT и автоматически настроить Эуклидос так, как это пожелает пользователь. Таким образом весь Эуклидол может быть сориентирован на тот или иной живой язык. В своих медитациях я часто буду заменять названия операторов русскими словами.

.1556. Итак, мы познакомились с такими структурными единицами языка Эуклидола, как знак, символ, предикат. Следующей еще более крупной структурной единицей текста является БЛОК – некоторое множество предикатов.

.1557. Каждый оператор однозначно характеризуется двумя показателями:

а) OBI (индекс блока);

б) OPI (индекс оператора в блоке).

.1558. Конкретные значения этих индексов для каждого оператора см. в автоскриптах Эуклидоса.

.1559. Как уже было сказано выше {.1535}, операторы по порядку следования в блоке подразделяются на:

а) отдельные;

б) головные;

в) промежуточные;

г) хвостовые.

.1560. Отдельные операторы сами представляют собой отдельный блок. Кроме того, блоком называются все операторы (предикаты) с одинаковым индексом OBI, следующие в потоке от головного до хвостового оператора (включительно). Операторы блока не обязательно должны идти подряд. В Эуклидоле допускаются вложенные блоки с произвольной глубиной вложенности, причем не имеет значения, перебивается ли описание одного блока блоком того же типа (тот же индекс OBI) или блоком другого типа. По окончании вложенного блока Эуклидос возвращается к временно прекращенному блоку. Другие способы чередования операторов разных блоков запрещены. Промежуточные и хвостовые операторы не должны появляться раньше, чем головной оператор.

.1561. Самой крупной структурной единицей в Эуклидоле является теория. Для Эуклидоса теория – это всё то, что поступило к нему на обработку за одно выполнение.

.1562. Дополнения к данному описанию смотрите в автоскриптах аппарата ПРЕДИКАТ.

9. Тетрадь ALGOR

АЛГОРИТМ
Аппарат работы с конкретными множествами в Эуклидосе

Nil actum credens, dum quid superesset agendum
Marcus Annaeus Lucanus
(Если что-то осталось доделать, то не сделано здесь ничего)

Написано: 1979.08 – 1980.05 Рига

Медия ALGOR (в Третьей Медиотеке медитация АЛГОРИТМ) содержит общее описание проекта первого аппарата Эуклидоса – аппарата работы с конкретными множествами.

1. Задачи аппарата АЛГОРИТМ

1980.04

.1563. Теперь настало время приступить к рассмотрению тех вещей, для которых, собственно, Эуклидос создан. В этой медитации будет в общих чертах рассмотрен первый рабочий аппарат Эуклидоса: аппарат АЛГОРИТМ – средства описания и интерпретации алгоритмов в Эуклидосе. Точное и исчерпывающее описание этого аппарата смотрите в автоскриптах Эуклидоса. С точки зрения языка Эуклидола здесь, соответственно, рассматриваются операторы, при помощи которых можно описать конкретные множества и манипуляции с ними.

.1564. Практического значения аппарат АЛГОРИТМ не имеет, то есть, практически нет необходимости действительно интерпретировать описанные средствами языка алгоритмы (достаточно того, что они описаны). Основная цель Эуклидоса – работа с абстрактными множествами. Но аппарат АЛГОРИТМ нужен для того, чтобы со всей отчетливостью видеть, откуда берутся эти абстрактные множества.

.1565. Итак, аппарат работы с конкретными множествами. Что же такое конкретное множество? Согласно принятой ранее концепции множества {.497}, множество существует в реальном мире, а в субъекте существует его номиналия. Эуклидос – это субъект, причем такой, в «голову» которого мы можем заглянуть. Что из себя представляет номиналия в Эуклидосе? Это структура типа «куст» {.1411}: ствол его образовывает запись COM – неоднородный массив фиксированной длины {.1336} (см. Фиг.2 {.1584}). В структуру COM входят указатели различных списков (списка элементов, списка имен, списка шефов, то есть тех множеств, которым данное множество принадлежит как элемент) и другие записи с различными сведениями о данном множестве.

.1566. Списки элементов, шефов и имен представляют собой структуры типа «бахрома» {.1412} – это списки стандартных элементов EMB, которые включают ссылки на другие структуры COM (в списках элементов и шефов) или на записи имен, которые хранятся в структурах TAMB.

.1567. Множество может быть непоименованным (в этом случае список имен пуст) или поименованным. В списках имен поименованных множеств может быть более одного имени. Первое имя обозначает множество как уникальный, единственный объект. Второе имя – это обозначение для любого из его элементов. Третье имя – обозначение для элементов этих элементов и т.д. Например, если в списке имеются четыре названия: «народы», «народ», «человек» и «орган», то это множество называется «народы», оно состоит из элементов, которые

называются «народ», любой элемент любого народа называется «человек», а любой человек в данной теории рассматривается как множество элементов, называемых «орган».

.1568. Итак: в реальном мире множество, а в Эуклидосе ему соответствует разветвленная структура, ядро которой составляет таблица СОМ. Зная адрес таблицы СОМ, можно просмотреть списки и найти, какие в этом множестве есть элементы (а элементами могут быть только другие таблицы СОМ), в каких множествах данная СОМ числится элементом, какие имена присвоены данному множеству в данной теории, а также получить из таблицы СОМ другие сведения о данном множестве.

.1569. Имеется также список ВЕХ всех существующих в данной теории конкретных множеств (см. Фиг.1 {.1583}).

.1570. Второй главной структурой в рассматриваемом аппарате Эуклидоса является алгоритм (структура АТМ). (См. Фиг.3 {.1585}). Это неоднородный файл неопределенной, неограниченной длины. В первую запись файла входят указатели списков имен, материалов и продуктов алгоритма. Эти списки – опять структуры типа «бахрома», их элементы ссылаются либо на таблицы АВМ (абстрактных множеств), либо на записи имен в структурах ТАМВ. Основную массу файла алгоритма составляют записи команд – это образования, очень похожие на команды в программах для ЭВМ.

.1571. Таблица абстрактного множества АВМ (см. Фиг.4 {.1586}) несколько похожа на таблицу СОМ – это неоднородный массив фиксированной длины. Она включает указатель списка имен, ссылку на алгоритм, а также записи с другой информацией.

.1572. Имена, запомненные Эуклидосом, хранятся в структурах ТАМВ (см. Фиг.5 {.1587}), которые связаны в список ВАМО. Структура ТАМВ состоит из двух частей: шапки блока и вектора имен (однородного массива переменной неограниченной длины). Одна таблица ТАМВ хранит однородные имена (либо имена конкретных множеств, либо алгоритмов и т.д.). Посредством списка ВАМО можно найти все названия, имеющиеся в данной теории, и в то же время можно пропустить блоки ТАМВ с ненужным типом имен.

.1573. Все имена, запомненные в этих структурах, должны быть уникальными и не должны повторяться независимо от типа имени, то есть от того, что это имя обозначает. Такие имена в Эуклидоле называются глобальными. Глобальными являются названия операторов, алгоритмов, абстрактных и конкретных множеств.

.1574. Некоторые имена Эуклидос запоминает временно и не в структурах ТАМВ, а в других структурах. Такие имена называются локальными. Локальными являются, например, имена переменных индексов и метки операторов. Локальные имена должны быть уникальными только внутри блока Эуклидола и внутри своего типа. Они могут совпадать с глобальными именами, но не с названиями операторов.

.1575. Теперь можно сформулировать задачи аппарата АЛГОРИТМ:

.1576. 1) В языке должны быть средства описания, а в Эуклидосе – построения исходного пространства элементов.

.1577. 2) В языке должны быть средства описания алгоритмов действий с исходными и ранее построенными элементами, а в Эуклидосе – транслятор, преобразовывающий это описание в структуру АТМ.

.1578. 3) В языке должны быть средства описания выборок (построения новых элементов из ранее существовавших по тому или иному алгоритму), а в Эуклидосе должен быть интерпретатор алгоритмов.

.1579. 4) В языке должны быть средства, при помощи которых можно приказать Эуклидосу распечатать (вывести) описания созданных им объектов (результатов), ну, а Эуклидос, естественно, должен уметь это сделать.

.1580. Таковы четыре части аппарата АЛГОРИТМ. Впредь я их буду называть так:

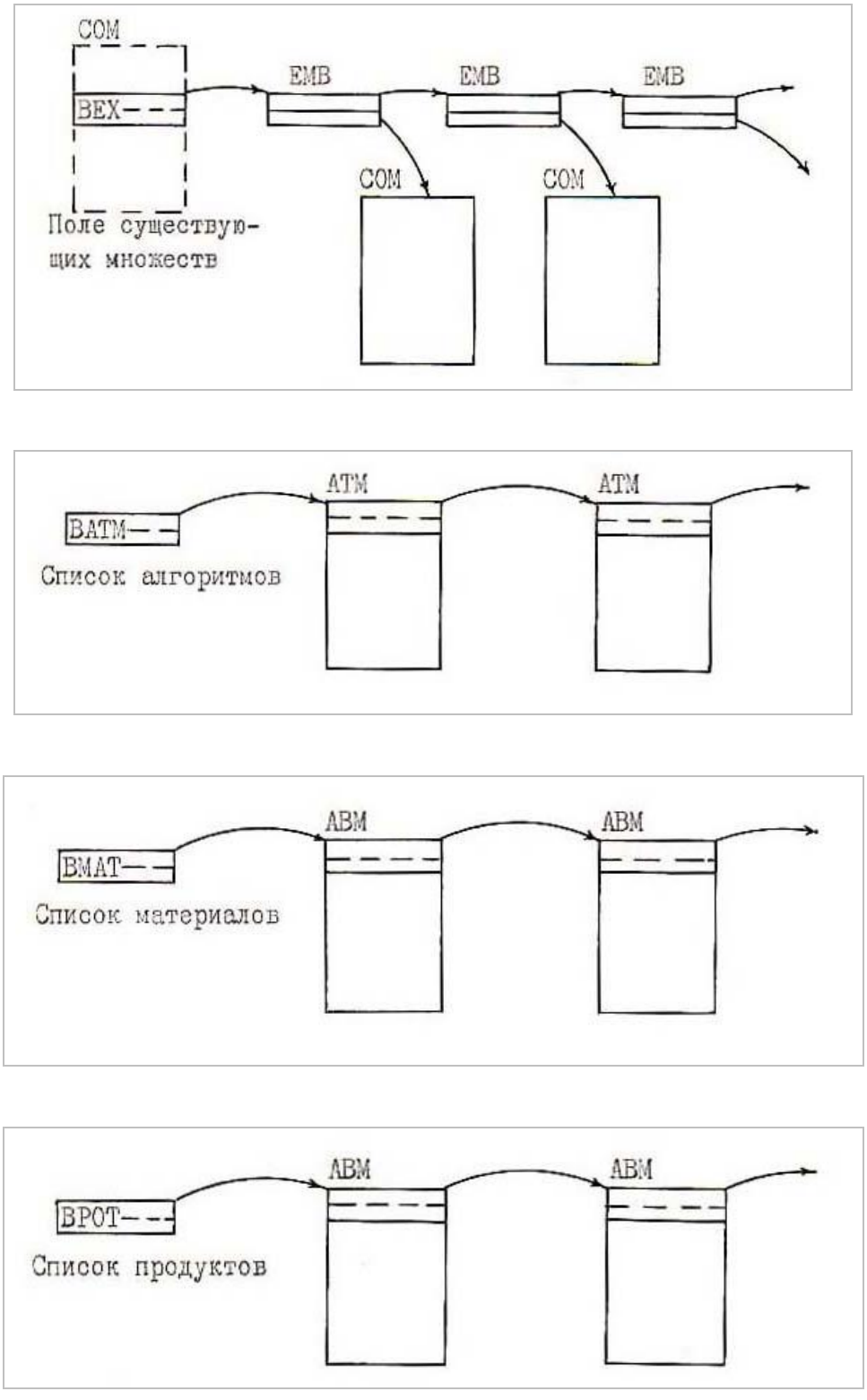
а) билдер;
б) транслятор;
в) интерпретатор;
г) корреспондент.

.1581. Итак, в общем работа с аппаратом АЛГОРИТМ ведется так: сначала автор теории (желающий познакомить со своим творением Эуклидоса) описывает исходное пространство объектов, а билдер строит их в «голове» Эуклидоса в виде таблиц СОМ. Потом (можно и сначала) автор описывает алгоритмы, какими он пользуется при выборках, а транслятор запоминает их в виде структур АТМ. Теперь автор говорит: «построим объект С из объекта В по

алгоритму А», а интерпретатор это делает в «голове» Эуклидоса. После этого автор может попросить корреспондента описать то, что там у Эуклидоса получилось (но может и не просить).

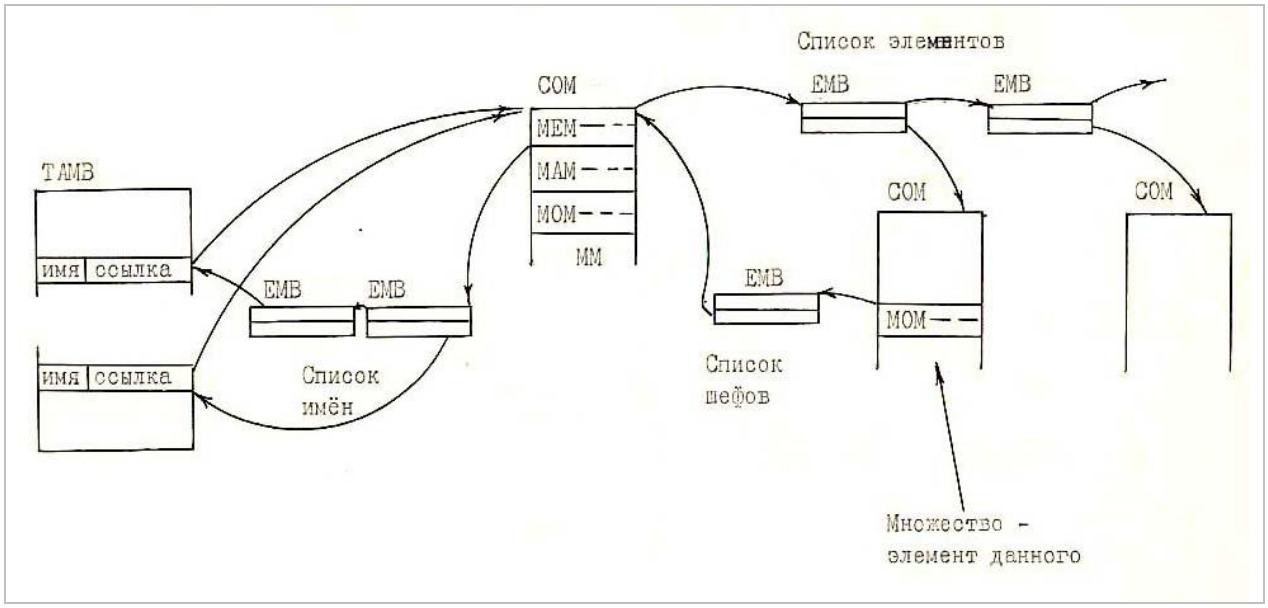
.1582. Такова сущность работы аппарата АЛГОРИТМ в Эуклидосе.

.1583.



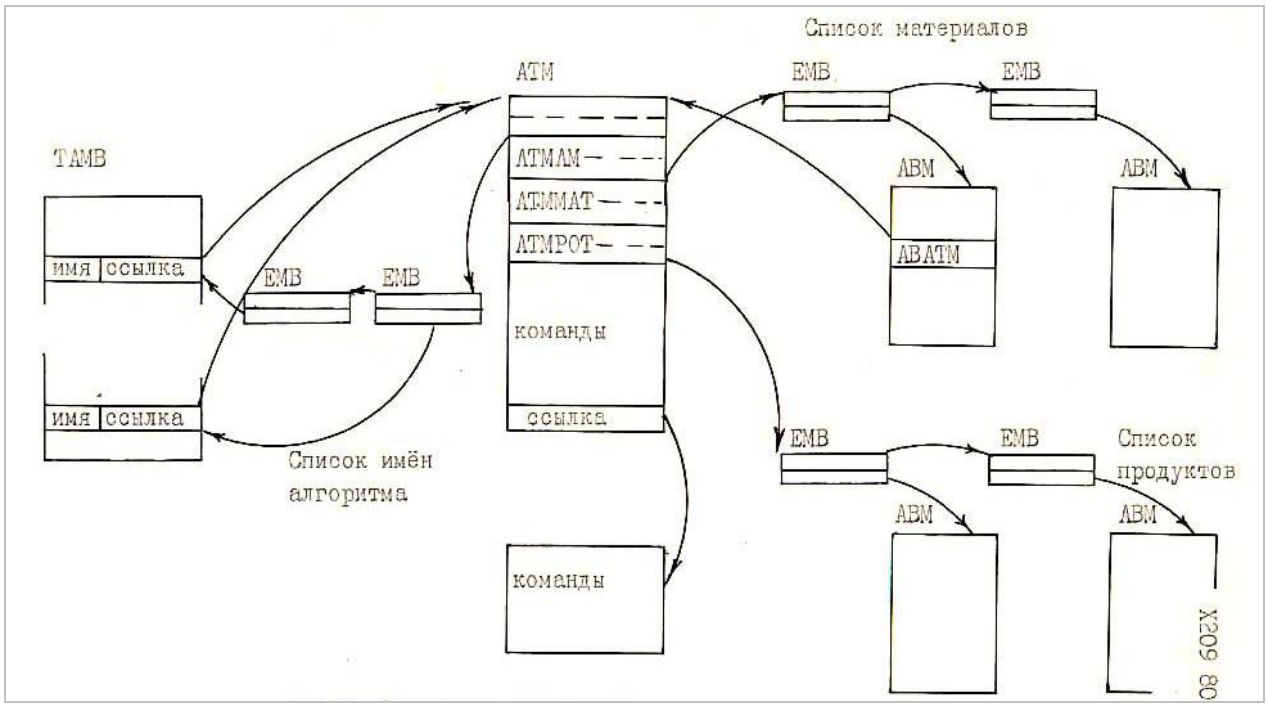
Фиг.1. Главные структуры Эуклидоса

.1584.



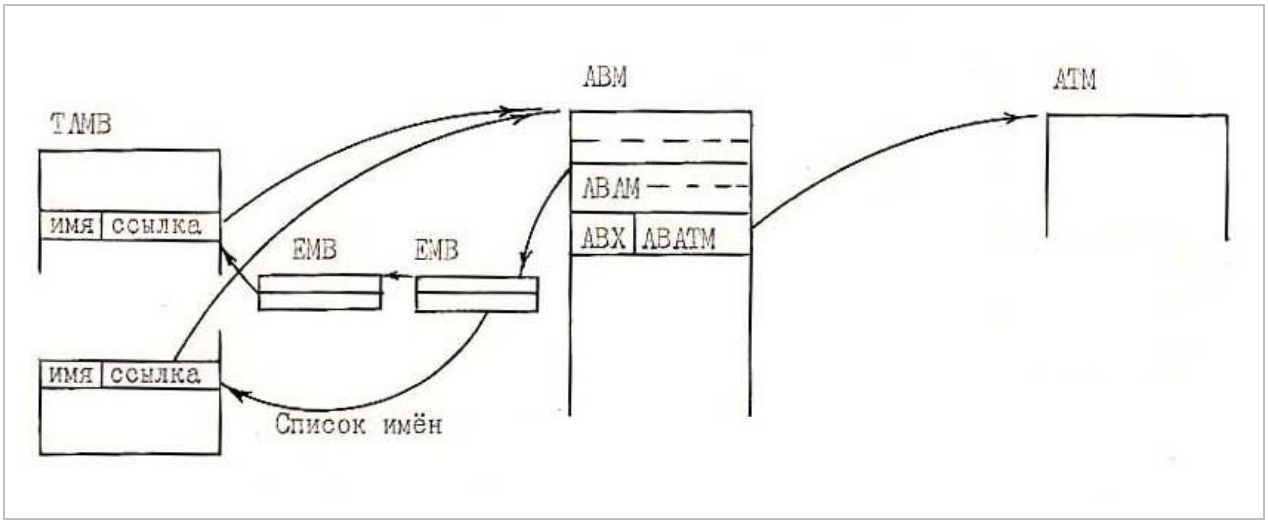
Фиг.2. Структура конкретного множества

.1585.



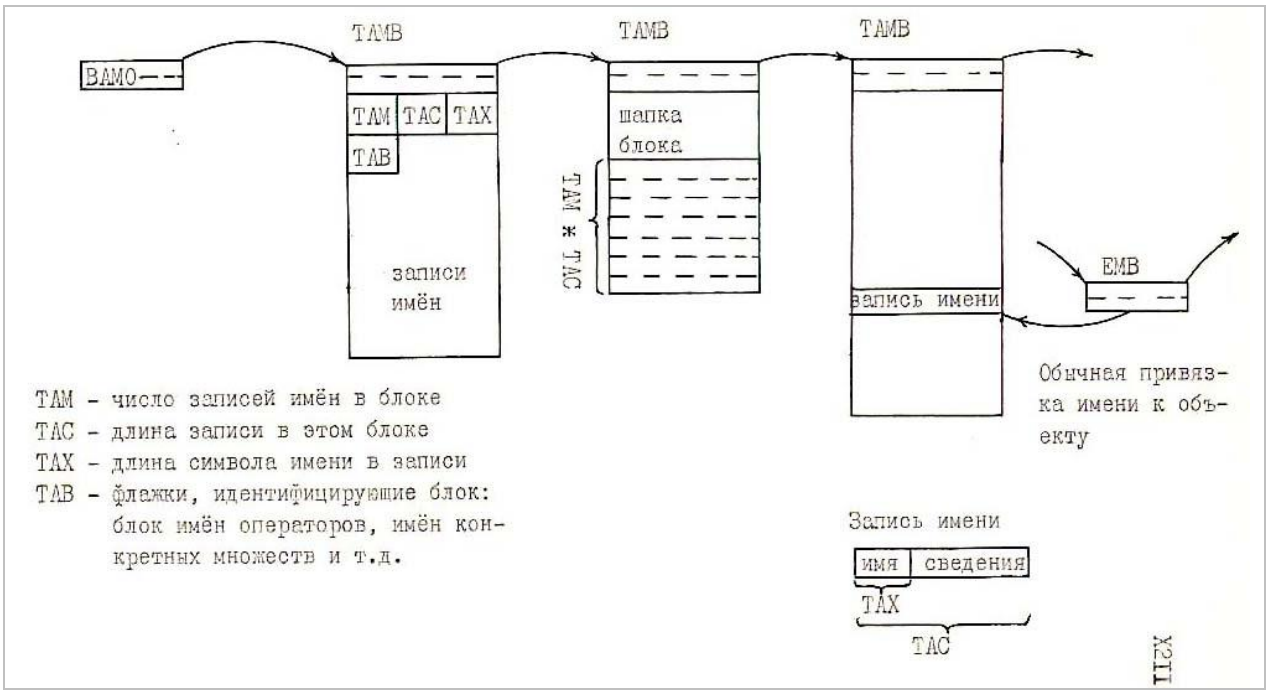
Фиг.3. Структура алгоритма

.1586.



Фиг.4. Структура абстрактного множества

.1587.



Фиг.5. Структуры запоминания имен

2. Билдер

1980.04

.1588. Главным оператором общения с билдером является ОПЕРАТОР EXT. У него три префикса и неограниченное количество суффиксов. Первым префиксом должна быть буква P,

вторым – знак равенства, а третьим – число. Эти три символа указывают уровень, на котором существуют указанные в суффиксах множества. К примеру запись

P=0 EXT A, B, C

.1589. означает, что на нулевом уровне существуют три объекта, называемые в Вашей теории именами A, B и C. Билдер построит три структуры COM и запомнит перечисленные имена.

.1590. Запись

P=1 EXT A(4)

.1591. означает, что Вы объявляете о существовании множества A из четырех непоименованных элементов. Билдер построит пять таблиц COM, причем четыре таблицы сделает элементами пятого множества.

.1592. Запись

P=3 EXT семья(родители(папа,мама),сыновья(3),дочери(2))

.1593. объявляет, что Вы в своей теории начинаете с множества (уровня 3), называемого «семья», которое состоит из трех элементов (уровня 2), называемых «родители», «сыновья» и «дочери». Эти элементы сами являются множествами, причем множество «родители» состоит из двух поименованных элементов (уровня 1), а множества «сыновья» и «дочери» состоят, соответственно, из трех и двух непоименованных элементов. То, что эти элементы являются объектами уровня 1, оставляет за Вами право когда-нибудь потом объявить, что сами они состоят из каких-то элементов. Если же Вы укажете в этом же операторе P=2, то множество «папа» (и другие) окажутся объектами нулевого уровня, то есть, точками, и Вы не сможете больше утверждать, что эти объекты сами состоят из каких-то элементов. Если же Вы в этом же операторе укажете P=1, то Эуклидос зафиксирует ошибку в Вашей теории.

.1594. Число операторов EXT не ограничено. Они могут появиться в любом месте теории. Число суффиксов в них тоже не ограничено. Глубина вложенности не ограничена (но должна быть согласована с уровнем, указанным в префиксах). Суффиксами и их подоперандами должны быть уникальные имена или числа. В первом случае создаются поименованные множества, а во втором – непоименованные.

.1595. Более подробное описание билдера см. в автоскриптах аппарата АЛГОРИТМ.

.1596. Вся работа системы Эуклидос на самом деле является моделью человеческой умственной деятельности, а именно: в основном по части дедуктивных выборок {.547}. Как известно, машинная реализация (и вообще моделирование) сенсорных, индуктивных и особенно перцептивных процессов представляется очень трудной, что означает, что алгоритмы этих выборок весьма сложны.

.1597. Билдер на самом деле является ничем иным, как аппаратом, позволяющим в Эуклидосе обойти все этапы сенсорных, перцептивных и индуктивных выборок и приступить сразу к дедуктивным выборкам.

3. Описания алгоритмов

1980.04

.1598. Алгоритм описывается рядом операторов, которые образовывают блок Эуклидола. Головным оператором блока является оператор HA, а хвостовым – оператор EM.

.1599. Оператор HA объявляет транслятору, что начинается описание нового алгоритма, и присваивает ему имя. Суффиксы и префиксы в операторе HA эквивалентны; если операндов несколько, то алгоритму присваиваются несколько альтернативных имен. Операторы

A HA B
HA A,B
A,B HA

.1600. эквивалентны, и все присвоят алгоритму имена A и B, по любому из которых Вы сможете сослаться на алгоритм. Обычно, конечно, достаточно одного имени.

.1601. Вторым оператором блока должен быть ОПЕРАТОР MAT, который описывает материал алгоритма, а третьим – ОПЕРАТОР PAT, описывающий продукты алгоритма. В блоке может быть только один оператор MAT и один оператор PAT. Формат у обоих операторов одинаковый: префиксы и суффиксы эквивалентны, операнды и подоперанды должны быть

уникальными именами и перечисляют абстрактные множества – соответственно материалы и продукты алгоритма, например, три оператора

HA сечения.
MAT секомое, секатор.
PAT перечисление, дополнение.

.1602. указывают, что у алгоритма «сечения» имеются два материала, называемые «секомое» и «секатор», и два продукта, называемые «перечисление» и «дополнение».

.1603. Если у операнда имеются подоперанды, то первый из них присваивает названия элементам множества, второй – элементам этих элементов и т.д., например, оператор

MAT человечество (человек, клетка).

.1604. объявляет, что материалом данного алгоритма является абстрактное множество, называемое «человечество», элемент этого множества называется «человек», а сами эти элементы тоже являются множествами и состоят из элементов, называемых «клетка». В дальнейшем при описании алгоритма можно сослаться на это множество и его элементы по любому из этих имен. С точки зрения транслятора эти имена эквивалентны, но они не эквивалентны с других точек зрения.

.1605. Число операндов и подоперандов в операторах MAT и PAT не ограничено, но общее число абстрактных множеств у одного алгоритма не должно превышать 16 (ограничение Эуклидоса).

.1606. При реализации алгоритма интерпретатором, ему вместо абстрактных множеств материала будут подставлены те или иные конкретные множества, а вместо абстрактных множеств продукта интерпретатор создаст какие-то конкретные множества.

.1607. Обычно имена, перечисленные в операторах MAT и PAT глобальны и не должны совпадать с глобальными именами каких-нибудь других объектов. Но если непосредственно перед именем идет знак равенства, то имя считается локальным и должно быть уникальным только в пределах блока.

.1608. После того, как объявлены названия материалов и продуктов, описывается сам алгоритм, то есть, какие-то манипуляции с теми конкретными множествами, которые при реализации алгоритма станут вместо абстрактных материалов и продуктов. При рассмотрении этих манипуляций нужно различать управление данными и управление программой {.1089}.

.1609. Данными у нас являются конкретные множества. Их представление в машине, то есть, язык данных в общих чертах я уже описал (это куст со стволом в таблице COM). Вопрос об управлении данными, это вопрос о том, как указать Эуклидосу (и вообще читателю теории) с каким именно объектом (какой из таблиц COM) нужно провести указанное действие.

.1610. Первый шаг в этом направлении – это указать имя абстрактного множества, названное в операндах операторов MAT и PAT. Но это множество состоит из многих элементов, причем обычно заранее не известно, сколько именно элементов будет в том конкретном множестве, которое во время реализации станет на место абстрактного множества.

.1611. Чтобы сослаться на элемент данного множества, используются индексы. Так запись

человек(3)

.1612. указывает на четвертый элемент (нумерация начинается с нуля!) во множестве «человечество», объявленное выше {.1603} оператором MAT (можно писать и «человечество(3)» и даже «клетка(3)», хотя речь идет о человеке). Такие индексы называются числовыми. Их можно использовать при описании алгоритмов, но большого значения они не имеют, так как сколько бы раз не выполнялось действие, всегда будет взят один и тот же объект.

.1613. Большее значение имеют переменные индексы. Запись

человек(a)

.1614. указывает на элемент во множестве «человечество», но на какой именно, зависит от того, какое числовое значение имеет переменный индекс «a» в данный момент, а это значение меняется Эуклидосом, и меняется таким образом, какой Вы укажете в описании алгоритма. Самое простое – прибавлять каждый раз к индексу единицу, таким образом Вы будете перебирать всё множество «человечество».

.1615. Можно указывать на элемент множества еще более сложным путем. Запись

человек(a+2*p)

.1616. делает ссылку на элемент с номером e = a + 2p. Эта ссылка уже зависит от значения двух переменных индексов в данный момент. Эуклидос допускает в подобном выражении не более трех членов (которыми могут быть имена переменных индексов и числа) и между ними до

двух арифметических операций (которыми могут быть сложение «+», вычитание «–», умножение «*» и деление «/»). Умножение и деление выполняется сначала, скобки не допускаются. Реально редко нужны более сложные выражения, чем выражения типа a+1.

.1617. Когда Вы произвели ссылку на элемент множества, этот элемент может быть опять множеством, на элементы которого нужно ссылаться. На эти элементы Вы можете указывать при помощи второго индекса (или индексного выражения), например,

человек (a+2*p, e+1).

.1618. Так можно продолжать неограниченно долго с любой глубиной индексации.

.1619. Этот принцип записи я использую не только в текстах Эуклидола (см. {.620}). Так, я буду обозначать различные множества символами длиной до 16 букв (обычно одной буквой). Если я пояснил, что такое множество A, то без особых оговорок и пояснений могу ссылаться на его элемент, употребив индекс, например, A(e). Запись A(к) обозначает тоже элемент множества A, но в общем случае другой. Записью A(e,p) я ссылаюсь на элемент множества A(e) и т.д. Индекс может принимать и конкретные значения, например: A(1), A(2), и относительные: A(e+1) – элемент, следующий за элементом A(e), и смешанные: A(3,e,к-2). Таким путем я ссылаюсь на один элемент, задаваемый «головным множеством» и одним или несколькими индексами.

.1620. Теперь рассмотрим управление программой, то есть, то, как Вы можете указать, в каком порядке нужно выполнять предписанные Вами манипуляции с данными. Обычно Эуклидос (интерпретатор) выполняет операторы (преобразованные в команды) в порядке их следования, и лишь изредка, встретив команду перехода (описанную оператором перехода) перескакивает на другое место алгоритма. Чтобы в операторе перехода указать, на какой оператор нужно перескочить, любому оператору, генерирующему команду интерпретатора, можно присвоить метку, а в операторе перехода сослаться при помощи этого идентификатора.

4. Описания команд

1980.04

.1621. Все описанные ниже операторы, включая оператор EM, генерируют команды интерпретатора. У всех у них может быть не более одного префикса. Если префикс присутствует, то это – метка оператора. Меткой может быть как имя, так и число и даже специальный символ. Метки должны быть уникальными внутри блока, но могут повторяться в разных блоках и могут совпадать с названиями других объектов (множеств, алгоритмов и т.д.), кроме названий самих операторов.

.1622. Основной набор действий, необходимых для построения одних множеств из других, был описан в {.638}. Исходя из этих соображений, построена система команд интерпретатора (соответственно операторы Эуклидола, описывающие эти действия). Ниже описан основной набор этих операторов. Полный их список и исчерпывающее описание см. в автоскриптах Эуклидоса.

.1623. Оператором XA можно увеличить значение переменного индекса на единицу, оператором XE уменьшить на единицу, оператором XO можно установить значение переменного индекса в ноль (на первый элемент множества), а оператором XEX можно установить значение одного переменного индекса равным значению второго переменного индекса. Операторы XA, XE и XO имеют один суффикс, а оператор XEX – два суффикса, которые все должны быть именами переменных индексов.

.1624. Переменный индекс можно обозначать любым именем, в том числе совпадающим с именами множеств, с метками и т.д., кроме названий операторов. Если в пределах одного блока дважды указывается одно и то же имя переменного индекса, то считается, что речь идет об одном и том же индексе. Если одно и то же имя указано в двух разных блоках, то считается, что речь идет о двух совершенно разных индексах.

.1625. Число всех употребляемых в одном блоке переменных индексов не должно превышать 16 (ограничение Эуклидоса). В исходном состоянии (перед любой реализацией алгоритма) значение переменного индекса равно нулю (указывает на первый элемент). Каждый раз, когда транслятор впервые встречает незнакомое ему имя индекса, он заводит новый переменный индекс. Поэтому ошибка при написании имени индекса может привести к появлению «фиктивного» переменного индекса.

.1626. Увеличение значения переменного индекса может привести к тому, что значение это станет больше, чем число элементов в том конкретном множестве, к которому Вы обращаетесь. В этом случае интерпретатор не выполнит предписанное командой действие.

.1627. Во всех командах, где производится обращение к множествам и их элементам, интерпретатор устанавливает индикатор признака результата, который может принимать два значения: E (существует) и O (не существует). Имеются команды (и операторы) перехода, которые передают управление разным операторам в зависимости от состояния этого индикатора. Таким образом Вы можете выполнять одни действия, пока элементы есть, и другие, когда элементы кончились.

.1628. Оператором TEX можно и непосредственно проверить, существует ли указанный суффиксом объект. Суффиксом должно быть имя множества (или его элементов), упомянутое в операторах MAT или PAT, а подоперанды образовывают (если они присутствуют) индексные выражения, как это было описано выше {.1615}. Такая конструкция (имя множества плюс, может быть, ряд индексов или индексных выражений) называется индексируемым операндом.

.1629. Оператором TEC можно проверить, принадлежит ли объект, указываемый первым индексируемым операндом, к множеству, указываемому вторым индексируемым операндом. В случае принадлежности устанавливается признак E, иначе признак O (в том числе и тогда, когда один или оба объекта не существуют).

.1630. Оператор BO генерирует команду перехода на оператор (команду), обозначенную меткой, которая указана суффиксом оператора. Если в момент выполнения команды установлен признак 0, то выполняется переход, в противном случае выполняется следующий оператор (команда).

.1631. Оператор BE действует аналогично, только переход осуществляется, если установлен признак E.

.1632. Оператор MO (включить в множество) имеет два индексируемых операнда. Первый индексируемый операнд указывает на то множество, к которому в качестве очередного элемента должен быть присоединен объект, указываемый вторым индексируемым операндом. Сначала проверяется, существует ли множество, указываемое первым операндом. Если оно не существует, создается его (пока пустая) номиналия. Потом проверяется, существует ли объект, указываемый вторым операндом. Если он не существует, устанавливается признак O, гасится признак E и выполнение оператора заканчивается. Если объект существует, устанавливается признак E, и объект присоединяется ко множеству.

.1633. Оператор OM (удалить из множества) имеет аналогичную структуру и смысл операндов, но производит обратное действие. Признак E устанавливается, если удаляемый элемент принадлежал данному множеству, во всех остальных случаях устанавливается признак O.

.1634. Оператор CE позволяет создать пустое множество как элемент какого-нибудь множества. Он имеет один индексируемый операнд, причем обязательно присутствие в скобках хотя бы одного индексного выражения. Последнее индексное выражение идентифицирует, в качестве какого элемента должно быть создано пустое множество в том множестве, которое указывается суффиксом и всеми предыдущими индексными выражениями. Например, оператор

CE A(e+1,к)

.1635. создаст пустое множество в качестве к-того элемента в множестве A(e+1). Если в этом множестве к-тый элемент уже существовал, то он отныне станет элементом к+1, и аналогично сдвинутся номера у всех дальнейших элементов.

.1636. Оператор EX (выполнить алгоритм) позволяет внутри реализации одного алгоритма выполнить другой с неограниченной глубиной вложенности. Первый суффикс должен быть именем алгоритма, который к моменту трансляции оператора EX уже полностью описан (был оператор EM). Это имя ранее было указано в операторе HA того алгоритма, который нужно выполнить. Дальше следует ряд индексируемых операндов. Объекты, обозначаемые индексируемыми операндами, будут поставлены выполняемому алгоритму в качестве материала или созданы им в качестве продуктов. Допускаются два формата операндов – позиционный и ключевой. В позиционном формате индексируемые операнды перечисляют конкретные множества, подключаемые вместо абстрактных в таком порядке, в каком абстрактные множества перечислялись в операторах MAT и PAT того алгоритма, который надлежит выполнить (к-тому материалу соответствует к-тый индексируемый позиционный операнд). В ключевом формате сначала указывается имя абстрактного множества, потом ставится знак равенства и

индексируемый операнд, указывающий на то конкретное множество, которое нужно подставить вместо данного абстрактного. Порядок ключевых операндов безразличен. В одном операторе EX можно применять оба формата, но тогда все позиционные операнды должны идти сначала, а потом все ключевые. Не важно, позиционным или ключевым способом, но надо указывать все материалы алгоритма. Продукты же можно указывать не все. Неназванные продукты не будут построены. Если множество, указанное как продукт алгоритма, уже существует, оно перед выполнением алгоритма будет разрушено, и по алгоритму построено заново. Если хотя бы один из объектов, перечисленных в операндах оператора EX и обозначающих материал, не существует, то алгоритм не выполняется и устанавливается признак O; если алгоритм выполнился, то, независимо от его результатов, устанавливается признак E.

.1637. Оператор EM заканчивает описание и трансляцию алгоритма, а команда, ему соответствующая, прекращает выполнение алгоритма. Суффиксы у этого оператора отсутствуют.

5. Трансляция и реализация

1980.04

.1638. На время трансляции одного алгоритма Эуклидос создает структуру TEB (см. Фиг.6 {.1650}). Она начинает строиться при поступлении оператора HA и разрушается при поступлении оператора EM. В списках этой структуры в виде буферов XBOP запоминаются различные сведения об алгоритме.

.1639. В списках названий алгоритма TEBAM запоминаются названия (из оператора HA) вместе с адресом структуры ATM, которая начинает строиться.

.1640. В списке материалов TEBMAT и в списке продуктов TEBPOT запоминаются имена из операторов MAT и PAT вместе с индексом множества. Первому операнду оператора MAT и всем его подоперандам присваивается индекс 0, второму – 4, третьему – 8 и т.д. Эта нумерация продолжается в операндах оператора PAT. Таким образом каждому имени сопоставлено число – номер множества, причем имена множества и его элементов имеют одинаковые номера.

.1641. В списке индексов TEBIX запоминаются имена переменных индексов, причем каждому новому имени присваиваются номера 0, 4, 8 и т.д.

.1642. В списке меток TEBKA запоминаются встретившиеся до сих пор метки вместе с адресом соответствующей команды в структуре ATM.

.1643. Транслятор аппарата АЛГОРИТМ является транслятором одноразового просмотра. Поэтому ссылки вперед он запоминает в списке TEBTO вместе с адресом того места в структуре ATM, куда надо проставить действительный адрес, соответствующий метке. Когда метка встречается и становится известен ее действительный адрес, он проставляется в нужное место, а соответствующие буфера из списка TEBTO удаляются. Поэтому обнаружить отсутствие метки транслятор может только при поступлении оператора EM: если в этот момент список TEBTO не пуст, значит имелись ссылки на несуществующие метки.

.1644. При поступлении очередного оператора транслятор генерирует команду по форматам, показанным в Фиг.7 {.1651} и 8 {.1652}. В первом байте любой команды находится код команды, который совпадает с кодом OPI соответствующего оператора (см. автоскрипты Эуклидоса). Символические названия множеств и переменных индексов преобразовываются транслятором при построении структуры ATM в однобайтовые номера, взятые из списков TEBMAT, TEBPOT и TEBIX, метки и названия алгоритмов – в абсолютные адреса соответствующих структур. Все индексные выражения, даже если они состоят из одного члена и не содержат знаков арифметических операций, всегда преобразовываются в четырехбайтовую структуру (Фиг.8 {.1652}) и при реализации алгоритма вычисляются интерпретатором одинаково.

.1645. Если при трансляции не были обнаружены ошибки, приводящие к абортированию алгоритма, то при поступлении оператора EM имена алгоритма заносятся в списки имен (структуры TAMB) и он становится доступным для выполнения, а по спискам материалов и продуктов строятся номиналии соответствующих абстрактных множеств. Во время трансляции и в случае ошибок алгоритм не доступен для выполнения, а абстрактные множества, им определяемые, отсутствуют. (Список ошибок и их характеристики см. в автоскриптах данного аппарата).

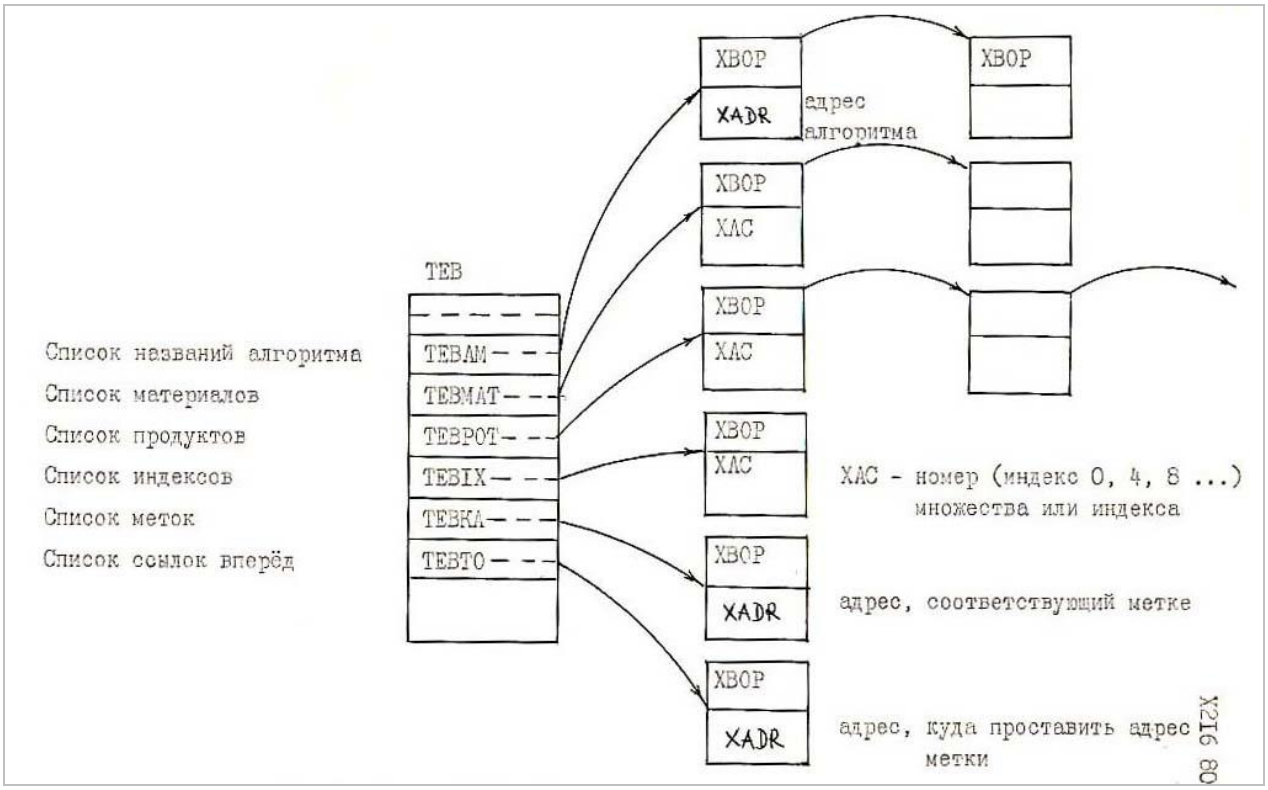
.1646. Обычно команды размещаются подряд в памяти машины, но трансляция одного алгоритма может быть перебита трансляцией другого алгоритма или каким-нибудь другим действием, требующим оперативной памяти. Тогда структура ATM будет состоять из нескольких участков, а в конце каждого из них (кроме последнего) транслятор сгенерирует команду безусловного перехода к следующему участку.

.1647. Реализация алгоритма может быть инициирована либо пользователем при помощи операторов управления интерпретатором, либо самим интерпретатором, когда он встречает команду EX. В любом случае вызывается интерпретатор (в случае команды EX это рекурсивный вызов) и ему поставляются (см. Фиг.9 {.1653}) адрес той структуры ATM (того алгоритма), который нужно выполнить, и блок множеств, содержащий адреса структур COM, представляющих конкретные материалы и продукты реализации. Интерпретатор заводит блок индексов, где хранит текущие значения индексов. Блок множеств и блок индексов – это стандартные буфера, длиной 64 байта, а адрес множества и значения индекса – четырехбайтовые слова (отсюда ограничение в 16 множеств и 16 индексов).

.1648. В процессе реализации алгоритма интерпретатор может и сам создавать структуры COM, являющиеся элементами его продуктов. Если реализация алгоритма вообще была инициирована, то продукты, названные в операторе EX, будут существовать всегда, в худшем случае они останутся пустыми множествами.

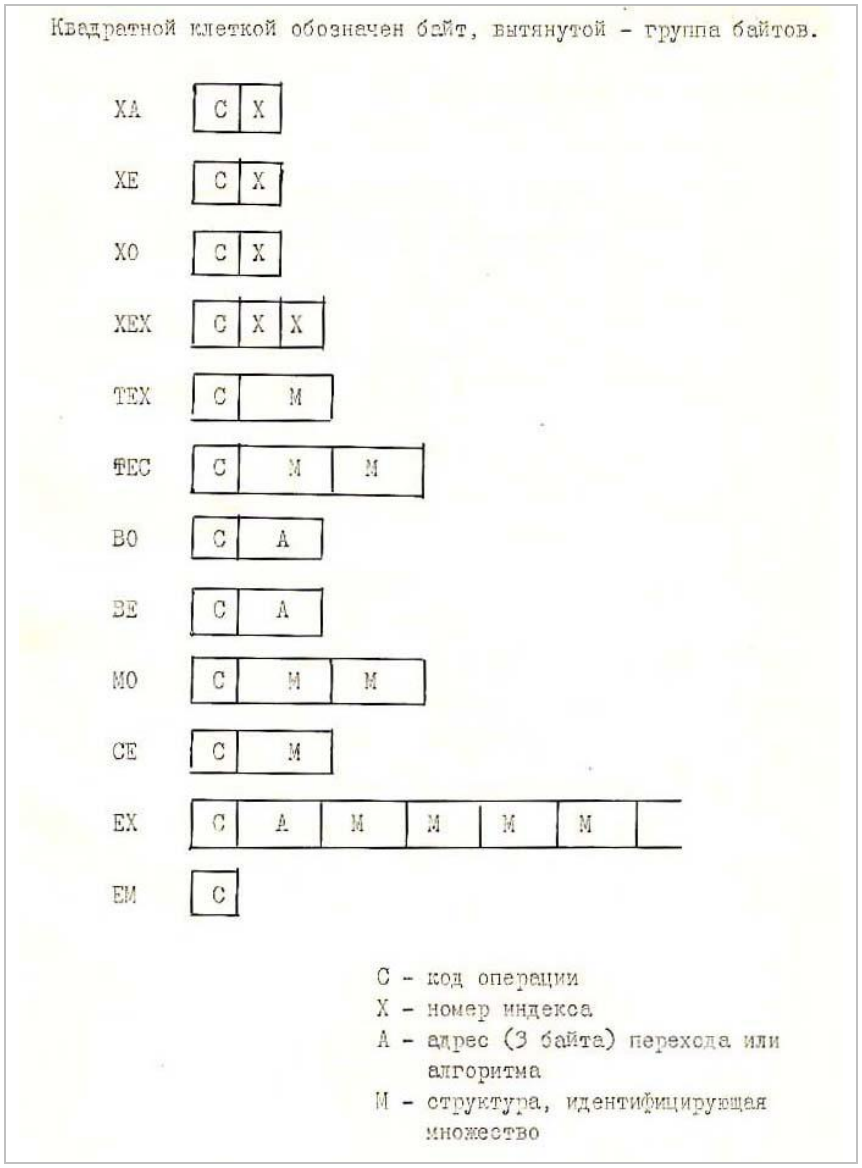
.1649. Продукты, которые не названы в операторе EX, интерпретатор фактически создает, но перед выходом опять разрушает.

.1650.



Фиг.6. Структура при трансляции

.1651.

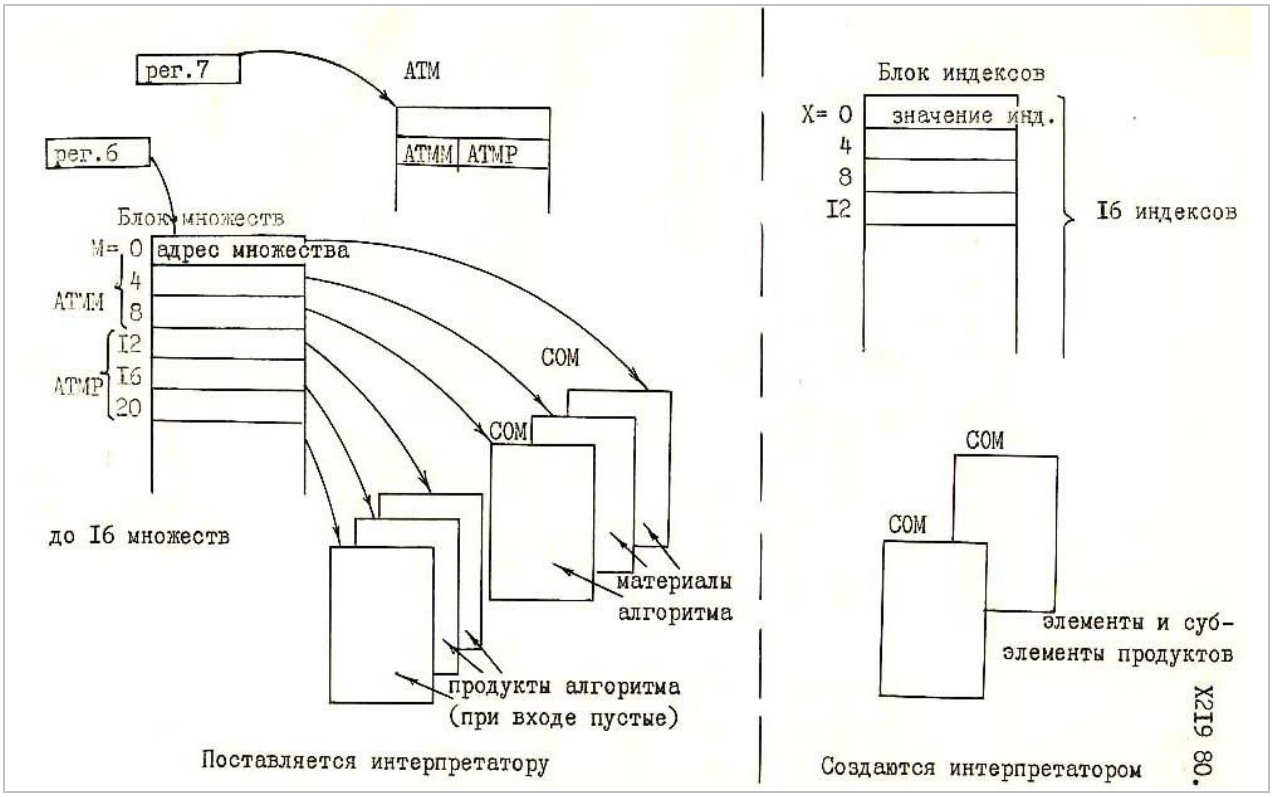


Фиг.7. Структура команд интерпретатора

.1652.

Фиг.8. Структура, идентифицирующая множество

.1653.



Фиг.9. Данные для интерпретатора

6. Примеры описания алгоритмов

1979.08
(раньше на 8 месяцев)

.1654. В качестве примеров записи алгоритмов на Эуклидоле я приведу описания тех пяти алгоритмов, которые в медитации ТЕОРИКА были изложены на словах. Вот описание алгоритма сечения {.646}:

```
§       НА      Сечения;        алгоритм называется алгоритмом сечения множества
                                другим множеством.
        МАТ     Секомое, Секатор;       одно исходное множество для этого алгоритма
                                называется секомым, другое – секатором.
        РАТ     Пересечение, Дополнение;        одно из создаваемых множеств называется
                                пересечением множеств, другое – дополнением
                                (секатора в секомом).
А1      ТЕС     Секомое(р),Секатор;     проверяем, имеется ли очередной элемент секомого
                                также и в секаторе.
        ВО      А2 ;    если нет, обходим помещение в пересечение.
        МО      Пересечение,секомое(р) ;        если да – помещаем элемент в создаваемое
                                множество пересечения.
        ВЕ      А3 ;                    и обходим помещение в дополнение.
А2      МО      Дополнение,Секомое(р)
А3      ХА      р ;                     переходим к следующему элементу.
        ТЕХ     Секомое(р) ;            проверяем, не исчерпано ли множество.
        ВЕ      А1 ;                    если нет, продолжаем.
        ЕМ ;            если исчерпано, то сечение сделано; из секомого множества сделаны два
                        таких, что всякий элемент секомого находится либо в пересечении, либо в
                        дополнении: обязательно в одном и только в одном из них, причем
                        разделение исходного множества сделано по признаку принадлежности
                        к другому множеству.            %
```

.1655. Этот фрагмент Эуклидола содержит очень много комментариев. То же самое можно сказать и лаконичнее:

§ НА сечения. МАТ секомое, секатор. РАТ пересечение, дополнение. А1 ТЕС секомое(р), секатор. ВО А2, МО пересечение, секомое(р), В А3, А2 МО дополнение, секомое(р). А3 ХА р ТЕХ секомое(р) ВЕ А1 ЕМ %

.1656. Разумеется, это другая крайность. Хотя по канонам Эуклидола это вполне допустимая запись (и Эуклидос ее разберет даже на ничтожные доли секунды быстрее, чем первую), но для удобства людей лучше всего размещать операторы в столбик, умеренно снабжая комментариями (длительные пояснения можно давать в полях: после знака процента и до знака параграфа). Вот пример такого описания {.650}:

```
§       НА      Объединения.
        МАТ     Сливаемое.
        РАТ     Объединение.
В1      ХО      р
В2      ТЕС     Объединение, Сливаемое(е,р)
        ВЕ      В3
        МО      Объединение, Сливаемое(е,р)
В3      ХА      р
        ВЕ      В2
        ХА      е
        ТЕХ     Сливаемое(е)
        ВЕ      В1
        ЕМ              %
```

.1657. В этом алгоритме по индексу е берется первый элемент сливаемого множества, рассматривается сам как множество и по индексу р все его элементы переносятся во множество объединения. Командой МО устанавливается признак Е до тех пор, пока не кончится множество Сливаемое(е). Оператор ХА признак не портит, и оператор ВЕ его проверяет: если множество исчерпано, модифицируется индекс е и переходим к следующему множеству – элементу Сливаемого. Одинаковые элементы командой МО не дублируются, так как в этом случае ее обходим.

.1658. Вот алгоритм, описанный на словах в главе ТЕОРИКА,27 {.701}:

```
§       НА      Произведения.
        MAT     Множимое (мультант), Множитель (мультатор).
        PAT     Произведение (мультон).
B1      XO      e
B2      XO      p
B3      MO      Произведение(a), Мультант(c,p).
        XA      p
        BE      B3
        XO      p
B4      MO      Произведение(a), Мультатор(e,p).
        XA      p
        BE      B4
        XA      a
        XA      e
        TEX     мультатор(e)
        BE      B2
        XA      c
        TEX     мультант(c)
        BE      B1
        EM              %
```

.1659. Вот два алгоритма из главы ТЕОРИКА,28 {.707} и {.721}:

```
§       НА      Развертки.
        MAT     Развертываемое.
        PAT     Развертка.
A1      MO      Развертка(a), Развертываемое(a)
BE      A1
        EM
        НА      Отбора.
        MAT     Выбор, Признак.
        PAT     Отбор, Выброс.
B1      TEC     Признак, Выбор(a)
        BO      B2
        MO      Отбор, Выбор(a)
        BE      B3
B2      MO      Выброс, Выбор(a)
B3      XA      a
        TEX     Выбор(a)
        BE      B1
        EM              %
```

.1660. В заключение еще один интересный алгоритм:

```
§       НА      Экспонирования.
        MAT     Экспобаза.
        PAT     Экспонента(Подмножество), Слои(слой).
        MO      Подмножество(e), Подмножество(e,p)
B1      XO      e
B2      MO      Слой(a,e), Экспобаза(a)
        BO      B5
        XO      p
B3      MO      Слой(a,e), Подмножество(e,p)
        XA      p
        BE      B3
        XA      e
        TEX     Подмножество(e)
```

```
        BE      B2
        XO      e
B4      MO      Экспонента, Слой(a,e)
        XA      e
        BE      B4
        XA      a
        BO      B1
        B5      EM      %
```

.1661. Первой командой этого алгоритма создается пустое множество как элемент экспоненты (если не верите, прочитайте внимательно описание оператора MO {.1632}). Экспонента (множество всех подмножеств) создается (см. предикат B4) последовательным вливанием в экспоненту ряда множеств, которые названы слоями. Слой p-того элемента – это множество всех тех подмножеств, которые можно построить первыми p элементами. Очередной слой образовывается присоединением ко всем уже построенным подмножествам очередного элемента, таким образом, каждый новый слой удваивает экспоненту. Если в экспобазе p элементов, то экспонента будет содержать 2^p элементов. Обратите внимание на то, что произойдет, если экспобаза будет пустой: экспонента будет состоять из одного пустого множества ($2^0 = 1$), а множество слоев – из одного пустого слоя (в общем случае число слоев равно p+1, причем пустой слой всегда образовывается последним).

.1662. Описание алгоритма на Эуклидоле – это самое точное его описание, какое вообще бывает в моих медитациях, а имена (названия алгоритмов и абстрактных множеств), перечисленные в операторах HA, MAT и PAT, я после этого считаю определенными самым точным образом, а себя вправе без каких-либо дополнительных пояснений пользоваться этими названиями и в обычном тексте. Поэтому читателю не следует пренебрежительно относиться к фрагментам, написанным на Эуклидоле. Если Вы не желаете углубиться в собственно алгоритм, то читайте хотя бы его шапку.

7. Об АЛГОРИТМЕ

1980.04
(через 8 месяцев)

.1663. Каждый, кто достаточно близко знаком с ЭВМ, конечно же, сразу увидел сходство Эуклидоса с ЭВМ: интерпретатор Эуклидоса – это не что иное, как программно реализованный процессор (машина), имеющая свою систему команд, а описание алгоритма на Эуклидосе – не что иное, как программа для этого процессора, написанная на языке программирования, напоминающем ассемблер. Поэтому я эту часть языка Эуклидола иногда буду называть ассемблером Эуклидола, а описания алгоритмов на Эуклидоле или на входном языке интерпретатора (языке структур ATM) именовать программами.

.1664. Что же это за машина, для которой мы тут пишем программы? Эуклидос – лишь модель. Настоящая машина – это процессор отражения, работающий в моем и в Вашем мозге.

.1665. Итак, наконец-то, после бесконечных страниц приготовлений, долгожданный язык Эуклидол начинает приобретать более менее определенные очертания. И что же оказалось? Оказалось, что он не имеет ничего похожего на традиционную символику математики, на кванторы логики предикатов и т.д. Этого ли Вы ожидали, мой читатель? Но ничего другого я Вам не обещал. Теория – это алгоритмы отражения, и лучше всего говорить о них на алгоритмическом языке, похожем на те алгоритмические языки, которые открыто себя именуют таковыми.

.1666. Вещи, которые в ТЕОРИКЕ Вам, читатель, наверно показались довольно расплывчатыми и неопределенными (номиналии, уровни, алгоритмы) здесь превратились в реальные объекты в виде таблиц COM и структур ATM в памяти машины, и запомненной в них информации. Именно для этой четкости и реальности мне и нужен был Эуклидос. Почти всё, что я говорю, осталось бы без Эуклидоса только туманными и пустыми фразами, недостойными Вашего внимания (по крайней мере в моих собственных глазах).

.1667. Что же касается самого интерпретатора алгоритмов в Эуклидосе, то, на мой взгляд, он не имеет никакого практического значения. Нет необходимости интерпретировать алгоритмы

с другой целью, как только для того, чтобы убедиться в правильности их описания. Нет проблем в создании интерпретатора – программы ЭВМ, которая над подставленными ей таблицами, обозначающими множества, проведет точно указанные алгоритмом операции. И вся суть и необходимость интерпретатора на самом деле заключается только в том, что его МОЖНО сделать, что алгоритм МОЖНО реализовать. Того, что алгоритм можно реализовать при помощи интерпретатора, нам всегда будет достаточно, мы будем описывать алгоритмы для того, чтобы изучать их свойства, а не для того, чтобы их действительно выполнять на машине.

.1668. Поэтому я не старался сделать интерпретатор и его систему команд таким, чтобы он работал как можно более эффективно. Я прекрасно понимаю, что то, что теперь делает целый алгоритм, иногда можно реализовать одной командой интерпретатора, работающей в десять раз быстрее. Но не в этом была моя цель. Наоборот, я старался свести все алгоритмы к минимуму элементарных действий (команд). Ни одну из описанных выше основных команд нельзя разложить на ряд остальных операторов (значит, они независимы друг от друга и элементарны). В принципе этого набора команд достаточно, чтобы описывать любые алгоритмы, состоящие из тех операций, которые оговорены в {.638}. Но для удобства описания и чтения алгоритмов я введу ряд неэлементарных операторов:

.1669. Оператор B (безусловный переход) – эквивалентен двум подряд идущим операторам BO и BE.

.1670. Оператор MOM (перенести из одного множества в другое множество). Оператор

```
MOM   A,B,C
```

.1671. (где A, B, C – индексируемые операнды) эквивалентен двум операторам

```
MO    A,B
OM    C,B.
```

.1672. Оператор MOA (поместить во множество все элементы). Оператор

```
MOA   A,B
```

эквивалентен следующей группе операторов:

```
A1    MO    A,B(a)
      XA    a
      BE    A1.
```

.1673. Оператор OMA (очистить множество). Оператор

```
OMA   A
```

эквивалентен группе:

```
A1    OM    A,A(a)
      XA    a
      BE    A1.
```

.1674. Оператор MOMA (перенести все элементы из одного множества в другое множество). Оператор

```
MOMA A,B
```

эквивалентен группе:

```
MOA   A,B
OMA   B.
```

.1675. В дальнейшем могут быть введены и другие расширенные операторы.

.1676. Такие расширенные операторы можно вводить в язык и динамически при помощи ОПЕРАТОРА MOPT. Операторы, вводимые по MOPT, имеют такой же статус, как оператор EX, то есть, это промежуточные операторы блока описания алгоритма. В структуре ATM по этим операторам генерируется команда EX, то есть – эти операторы фактически являются расширениями оператора EX.

.1677. Префиксы и суффиксы у оператора MOPT эквивалентны. Операнды указывают имя нового оператора, а их подоперанды – имя алгоритма, например, оператор

```
MOPT  объединить (объединения)
```

.1678. введет новый оператор ОБЪЕДИНИТЬ, который будет заключаться в выполнении алгоритма объединения. У этого оператора префикс является меткой, а суффиксы определяют материалы и продукты алгоритма аналогично тому, как это делается в операторе EX.

.1679. Действие, аналогичное действию оператора MOPT, можно вызвать и другим способом: указав название нового оператора в подоперандах оператора HA, например, оператор

НА Объединения (Объединить)

.1680. выполнит помимо своих основных функций также и действия указанного выше оператора МОРТ.

8. Вызов интерпретатора и корреспондента

1980.04

.1681. Теперь осталось только одно: выяснить, как можно вызвать интерпретатор и как можно убедиться в том, что Эуклидос действительно выполнил то, что Вы ему предписали (и в том, что Вы написали в своем описании алгоритма именно то, что хотели написать).

.1682. Интерпретатор вызывается двумя главными операторами: EXEC и ECT.

.1683. У оператора EXEC префиксы и суффиксы равноправны; первый операнд должен быть именем алгоритма, остальные в позиционном или ключевом формате перечисляют материалы (все) и продукты (можно лишь часть) алгоритма аналогично тому, как это делалось в операторе EX, например:

Сечения EXEC A,B, перечисление=C, дополнение=E.

.1684. Надо учитывать, что множества, сопряженные с материалами (в примере: A и B) должны уже существовать, а имена (C и E) продуктам будут присвоены и не должны совпадать с уже задействованными именами.

.1685. Оператор ECT имеет один префикс и один суффикс, у которого столько подоперандов, сколько материалов имеется у данного алгоритма, например:

C ECT перечисление(A,B).

.1686. Явно название алгоритма здесь не присутствует. Эуклидос найдет нужный алгоритм по имени абстрактного множества «пересечение». Подоперанды перечисляют конкретные материалы алгоритма в таком порядке, в каком они определены в операторе MAT данного алгоритма. Имя C будет присвоено продукту, соответствующему тому абстрактному множеству, которое названо в суффиксе, а второй продукт (в общем случае все остальные) будут построены и тут же разрушены. Оператором ECT можно построить и назвать только один продукт алгоритма.

.1687. Оператор PEO имеет смысл и формат, аналогичный оператору EXEC. Разница между ними заключается в том, что по оператору PEO вызывается не настоящий интерпретатор алгоритма, а псевдоинтерпретатор. Он строит номиналию потенциального множества (структуру POM). Это псевдоконкретное множество: вместо того, чтобы на самом деле совершить все шаги, предписанные алгоритмом, и действительно построить конкретное множество, псевдоинтерпретатор беззастенчиво объявляет, что он уже всё сделал, сразу строит номиналию и запоминает в ней, каким продуктом какого алгоритма и от какого материала это якобы конкретное множество является. Псевдоинтерпретатор может «реализовать» и такие алгоритмы, которые для интерпретатора нереализуемы.

.1688. Узнать о том, какие конкретные множества Эуклидос построил, можно при помощи ОПЕРАТОРА РУ. У него префиксы и суффиксы эквивалентны, он может иметь любое число (в том числе 0) индексируемых операндов. Если операнды отсутствуют, Эуклидос выводит список всех построенных им конкретных множеств, иначе он распечатывает список элементов тех множеств, которые указаны в индексируемых операндах, например, оператором

РУ A,B(4)

.1689. Вы можете заказать список элементов множества A и множества, которое числится пятым по счету (четвертым по номеру) элементом множества B.

.1690. Таков в основных чертах аппарат АЛГОРИТМ работы с конкретными множествами в системе Эуклидос.

.1691. Я сохраняю за собой право в любой момент пополнять его в любом направлении, не оговаривая это в настоящей работе. Поэтому требуйте от Эуклидоса его самоописание, автоскрипт аппарата АЛГОРИТМ как продолжение этой медитации.

(Продолжение сборника «О природе чисел» в {R-NATUR3})

Оглавление