КАЛЛИСТО

КЛАССИК

Каллисто Классик

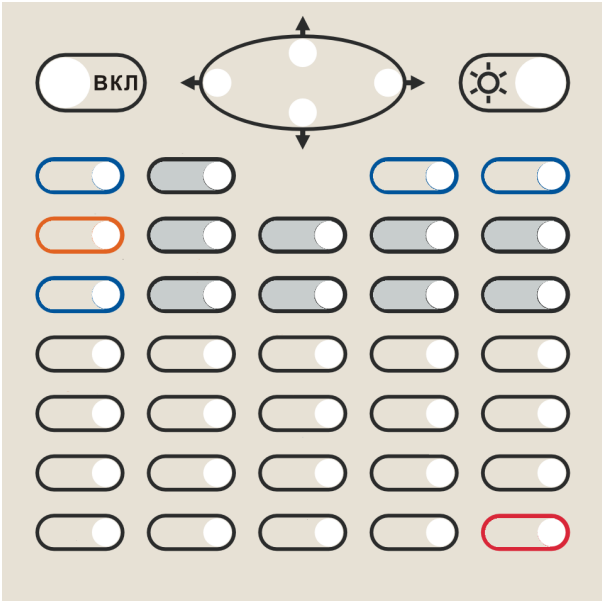
Алгоритмический язык программирования для «Электроники МК-161»

Руководство по эксплуатации

(C) 2016, 2017
Илья Васильев
Михаил Рамендик

Распространяется по лицензии GNU GPL версии 3 или более поздней
Использование примеров кода на Каллисто разрешается без ограничений

Москва-Лимерик, 2017



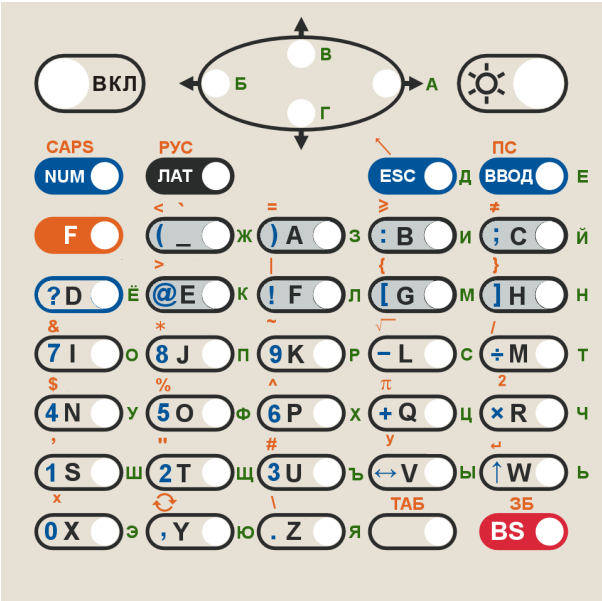


Рис.1 *Оверлей* — накладная клавиатура для Каллисто на «Электронике МК-161» в натуральную величину.

Внимание: для распечатки клавиатуры из файла PDF в правильном размере необходимо проверить настройки печати файла. Должен быть установлен реальный размер (100%). Уменьшение при превышении максимального размера (shrink to page) должно быть отключено. Некоторые программы не позволяют отключить такое уменьшение; авторы проверяли, что его можно отключить в официальном Adobe Reader для Windows и в evince (программе Gnome для просмотра документов) для GNU/Linux.

Оглавление

1 Каллисто....5
1.1 Краткая история....5
1.2 От разработчика....6
2 Комплект поставки Каллисто....8
3 Установка Каллисто....9
3.1 Установка Каллисто на калькулятор....9
3.2 Установка Каллисто на эмулятор калькулятора....10
4 Клавиатура калькулятора для Каллисто....11
5 Введение в язык Форт, диалект Каллисто....12
5.1 Программирование на Форте: слова....12
5.2 Работа с числами: стек и математические операции....14
5.3 Стековая нотация слов....15
5.4 Работа с данными: переменные, величины, константы....16
5.5 Условный оператор....18
5.6 Цикл FOR со счётчиком....19
5.7 Циклы BEGIN....20
5.8 Выделение памяти, работа со строками и массивами....22
5.9 Слова для порождения новых слов....24
6 Разработка и сохранение приложений....26
6.1 Сохранение состояния Каллисто....26
6.2 Хранение исходного текста приложения на внутреннем диске ЭКВМ....26
7 Блоки, хранение исходного кода и его компиляция....27
7.1 Перечень блоков....27
8 Строковый редактор EDIT....28
9 Главные отличия Каллисто от стандартного Форта....31
9.1 Стандартные слова Каллисто, которые могут быть плохо знакомы новичкам....31
9.2 Слова, существующие только на Каллисто....32
9.3 Таблица слов Каллисто, не имеющих аналогов в Форте....32
10 Сообщения об ошибках....33
11 Внутренние и дополнительные возможности Каллисто....34
11.1 Ввод и вывод....34
11.2 Машинная графика....34
11.3 Как Каллисто использует регистры МК-161....35
11.4 Единое адресное пространство....36
12 Хранение чисел в памяти....37
13 Словарная статья....37
14 Решение проблем....38
15 Синтаксис Каллисто Классик в РБНФ....39

1 Каллисто

Каллисто — это алгоритмический язык, компактный диалект языка Форт для калькуляторов серии «Электроника МК-161», «Электроника МК-152» и совместимых с ними. Среда Каллисто позволяет программировать на языке высокого уровня непосредственно «на борту» «Электроники МК-161», то есть без необходимости приобретения зарубежных компьютеров и использования внешних сред разработки.

Каллисто разработана при помощи *расширенного входного языка калькулятора* (ЯМК).

Каллисто создана в Москве на основе Форта и его наследника colorForth. Каллисто распространяется по свободной лицензии GPL версии 3, которую можно найти в каталоге gpl на русском и английском языках.

Подробную информацию о Каллисто можно найти на вики по адресу pmk.the-hacker.ru

1.1 Краткая история

Каллисто названа так в честь четвёртой луны системы Юпитера — самого дальнего из четырёх крупнейших спутников самой большой планеты нашей Солнечной системы, открытых в 1610 году. Иногда её даже называют Юпитер IV.

С Земли Юпитер (♃) хорошо виден невооружённым глазом, как яркая оранжевая звезда. В телескоп гигантский Юпитер виден как диск, по которому можно изучать характерные детали его атмосферы. Но Юпитер представляет собой газовый гигант, на котором даже нет поверхности, чтобы высадиться. Каллисто легко увидеть в хороший бинокль и на ней есть не только твёрдая поверхность, но и какая-никакая гравитация. Человек, когда туда доберётся, сможет ходить по ней. Каллисто побольше нашей Луны и незначительно меньше Меркурия. Каллисто — самый перспективный спутник Юпитера для колонизации, т.к. он единственный из четырёх крупных спутников находится за пределами смертельного радиационного пояса Юпитера.

Первоначально (с августа 2014 по март 2015 года) Каллисто планировалась как педантичная реализация готовящегося стандарта Форта-2012. Однако реальность показала, что подобный подход утопичен.

Первоначальные версии Форта были разработаны на весьма маломощных компьютерах и позволяли использовать ограниченные ресурсы с максимальной эффективностью. Стандарты языка, однако, появились позже, когда компьютеры стали заметно мощнее и «подстраивать» язык под особенности конкретной машины уже не было необходимости. Эти стандарты позволили писать на Форте *переносимые* программы, работающие на любом компьютере. Но написание таких программ может занимать больше времени, а их исполнение требует больших вычислительных ресурсов, чем когда слова Форта и оборудование традиционно подстраивают друг под друга.

«Электроника», однако, сходна по производительности и доступным ресурсам именно с компьютерами времён раннего Форта — особенно с учётом издержек реализации языка высокого уровня на ЯМК. Поэтому разработчик Каллисто поступил традиционным для Форта путём. Сейчас архитектура Каллисто отражает архитектуру «Электроники МК-161». Версия языка Форт, учитывающая стандарты, но разработанная специально для «Электроники», позволяет удобно управлять машиной с максимальной доступной скоростью.

Тем не менее, при возможности всегда применялись решения, стандартные для Форта. Однако, когда для обеспечения совместимости с зарубежным стандартом пришлось бы потерять в быстродействии или эффективности использования памяти, совместимостью приходилось жертвовать.

Кроме того, версия Форта, реализованная в Каллисто, вобрала в себя лучшее из входного языка (ЯМК), набора символов и клавиш «Электроники». «Перелёт на Каллисто» осознанно сделан максимально безболезненным и даже удобным для тех, кто знаком с програм-

мированием «Электроники МК-161» и советских ПМК. При этом возможности программиста на Каллисто существенно шире, чем на ЯМК.

Перенос существующих программ с Форта на Каллисто возможен. Но для этого программисту придётся учесть не только различие в версиях языка, но и особенности «Электроники МК-161». В частности, следует обратить внимание на скромные размеры её индикатора и памяти, наборы клавиш и символов. Особенности Каллисто будут требовать от программиста внимательности при переводе программ, по мере работы над ними приобретающих характерный внешний вид программ для «Электроники». Более того, в существующей версии весь текст программы потребуется ввести с алфавитно-цифровой клавиатуры калькулятора.

Такие проекты можно сравнить с доставкой земных товаров на небесное тело Каллисто. Впрочем, затраты это не такие уж космические и доступны как профессионалам, так и наиболее опытным любителям.

Добро пожаловать на Каллисто!

1.2 От разработчика

Транслятор Каллисто Классик было бы невозможно успешно разработать и представить на ваш суд, не опирайся я на успех гигантов. Вместе с релизом говорю «спасибо» моим предшественникам и коллегам с разных концов нашей планеты.

В первую очередь достоин благодарности Чарльз Мур, выпускник МТИ, разработавший для своей профессиональной деятельности замечательный язык Форт. В этот язык внесли и до сих пор вносят свой вклад множество людей с разных концов планеты, включая комиссию по разработке нового стандарта Forth 200x под руководством Антона Эртла из Венского технического университета. Далеко не всё из будущего стандарта было реализовано в первой версии Каллисто, но я с живым интересом слежу за их успехами и предложениями.

Слово «Классик» в названии языка отражает задачу сохранения и перевода на современную отечественную базу достижений двух популярных советских реализаций Форта — ФОРТа-ЕС, разработанного в ЛГУ под руководством профессора Сергея Николаевича Баранова, а также московского Форта ИТЭФ, реализованного Юрием Алексеевичем Семёновым из МФТИ для ПК фирмы IBM. Советскую литературу по этим двум системам можно использовать при изучении Каллисто наряду с данным Руководством.

Оборудование для Каллисто и язык программирования, на котором написана Каллисто Классик, основываются на советских разработках киевского НПО «Кристалл». Среди разработчиков первых отечественных ПМК наиболее известен Ярослав Карпович Трохименко, профессор НТУУ КПИ. Через десятки лет наша любимая архитектура была продолжена коллективом Новосибирска, из которых сообществу известно имя главного инженера Михаила Борисовича Степанищева, долгое время помогавшего народу осваивать «Электронику МК-161».

Движение за свободное программное обеспечение, помогающее успешно разрабатывать совместные проекты людям разных стран и убеждений, было основано Ричардом Столмэном из МТИ — того же института, что и Чарльз Мур. Каллисто использует лицензию GNU GPL v3, составленную под руководством Ричарда.

Экспериментальное описание синтаксиса Каллисто в форме РБНФ появилось как результат моих встреч в Москве и Нижнем Новгороде с профессором Никлаусом Виртом, разработчиком языков Паскаль и Оберон. Эти формальные правила пока не претендуют на полноту или абсолютную точность, но должны помочь получить начальное представление о Каллисто. Также Оберон повлиял на название предложенного вам языка. Много достижений Оберона ждёт своей реализации в последующих версиях Каллисто.

Идея Каллисто родилась в результате анализа критики «Электроники МК-161» и изучения пожеланий владельцев этих ЭКВМ. Эта работа проводилась как на сайте «Кон-Тики», так и в крупных социальных сетях. После многолетних и не всегда плодотворных бесед с разработчиками было решено самим сделать то, что нам нужно или просто хотелось бы видеть в ПМК — «бесконечный» стек, имена переменных и т.п. Выпуск Каллисто Классик завершает первый этап на этом пути, но было бы неправильным скрывать от читателя имеющиеся

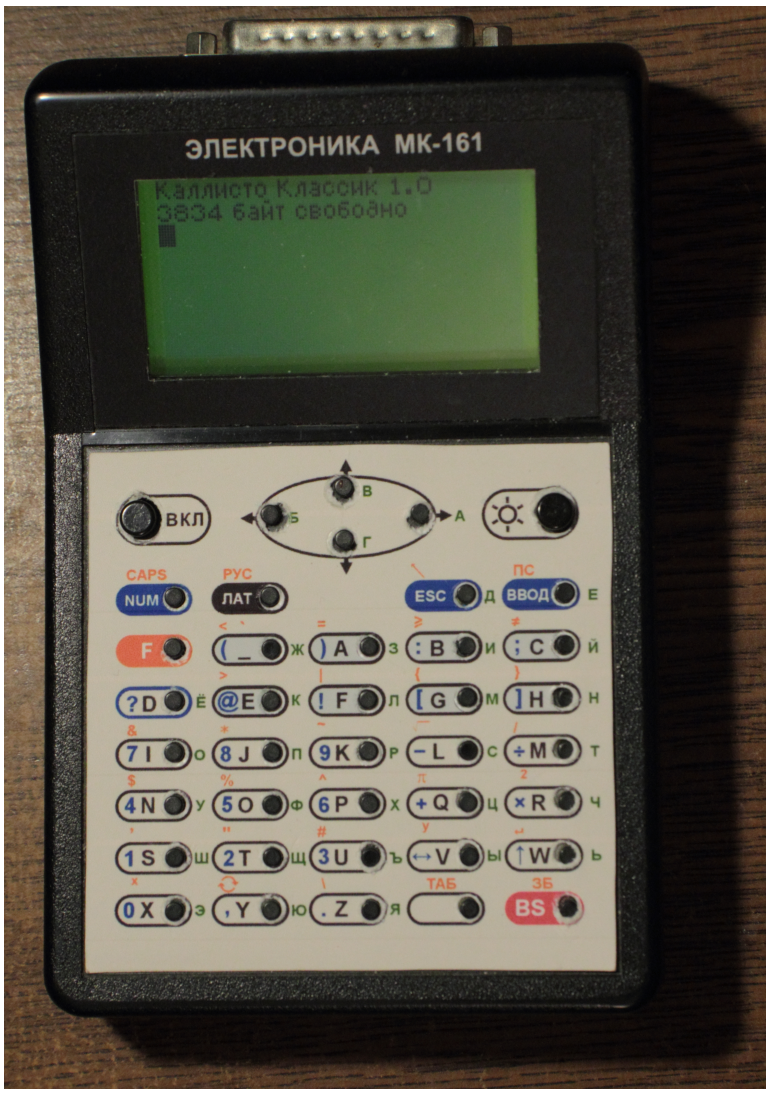
недостатки. По быстродействию Каллисто Классик значительно отстаёт от входного языка МК-161, хотя и превосходит советские ПМК в 10-20 раз. Быстрая загрузка сохранённого состояния отчасти компенсирует неприлично долгое, многоминутное время компиляции «на борту» (эмулятор МК-161 компилирует быстрее) — хотя сама возможность такой компиляции вдали от компьютера уже есть важное продвижение вперёд. Разработан подход, который позволит несколько улучшить эти характеристики в следующей версии Каллисто. Также входящий в комплектацию редактор (см. главу 8) весьма скромен и имеет лишь минимальный функционал. Каллисто нуждается в герое, который возьмёт на себя написание удобного редактора.

Рисунок обложки Руководства взял на себя Manwe из демо-группы SandS, а само Руководство — многомесячный профессиональный труд Михаила Рамендика, моего друга детства по Московскому Городскому Дворцу Пионеров и Школьников. За соседними компьютерами в 1980'ых годах мы вместе изучали программирование на превосходных компьютерах японского стандарта MSX, выпущенных фирмой «Ямаха». Эти компьютеры идеально подходили для обучения, определили наш жизненный путь. Все мы вспоминаем с теплотой и сами эти машины, и, разумеется, преподавателей, многое сделавших для того, чтобы мы могли познакомиться с компьютером действительно близко.

Пожелаю, чтобы Каллисто принесла вам столько же радости и оставила в вашем сердце такое же тёплое впечатление, какие дало нам наше счастливое советское детство в компьютерном классе «Ямах» и домашние посиделки с родными советскими ПМК.

Счастливого программирования на Каллисто,

Васильев И.В., г. Москва



2 Комплект поставки Каллисто

Каллисто распространяется в архиве `ZIP`, название которого содержит номер версии.

Запускаемым файлом Каллисто является файл `Callisto.mkp` . Этот файл устанавливается на калькулятор или эмулятор калькулятора.

Исходный текст Каллисто располагается в файле `Callisto.mkl` в кодировке Windows. При желании вы можете самостоятельно скомпилировать Каллисто стандартными средствами в уже известный вам исполняемый файл `Callisto.mkp`. Исходный текст транслятора крайне полезен для глубокого изучения Каллисто.

В каталоге `Blocks-source` содержатся блоки — полезные инструменты и примеры, написанные для «Электроники МК-161» на языке Каллисто. Например, файл `Blocks-source/B011.mkt` содержит исходный код *строкового редактора EDIT* — стандартное средство Каллисто для написания и редактирования программ.

Каталог `Blocks-compiled` содержит эти же блоки, заранее откомпилированные поставляемой версией Каллисто.

В каталоге `kbd` вы найдёте графические файлы с накладной клавиатурой для калькулятора и эмулятора калькулятора.

В каталоге `scrshot` расположены снимки экрана «Электроники МК-161», на которой работает данная версия Каллисто. Помимо графических файлов в нём находятся файлы с расширением mki, содержащие снимки экранов в заводском формате ЭКВМ (калькулятора).

В каталоге `tools` расположен исходный текст полезных программ-инструментов на языке Си. Например, программа `listmkt` позволяет перевести программы из формата буфера ЭКВМ в текстовый формат для просмотра на компьютере. Для запуска этих программ потребуется любой компилятор с языка Си.

3 Установка Каллисто

3.1 Установка Каллисто на калькулятор

Каллисто использует клавиатуру калькулятора для ввода русских и английских букв. Для удобства работы лучше всего распечатать клавиатуру (см. Рис. 1) и наложить её на корпус калькулятора, прорезав отверстия для клавиш. Подробнее о клавиатуре рассказано в главе 4 «Клавиатура калькулятора для Каллисто».

После успешной установки Каллисто калькулятор будет автоматически запускать Каллисто при включении. Это можно сравнить с установкой операционной системы на обычную ЭВМ. Если запуск не произошёл, нажмите клавиши `В/О С/П` для «ручного» пуска Каллисто.

Внутри Каллисто вы будете писать программы, из неё же их запускать. Снимать накладную клавиатуру вам придётся редко, только при выходе из Каллисто для составления и запуска программ на языке МК (команда `BYE`).

К счастью, процесс установки Каллисто проще и быстрее, чем установка операционной системы. Если вы передавали на МК-161 с компьютера хотя бы одну программу, проблем с установкой Каллисто у вас не возникнет.

Передача программ и данных на калькулятор производится программой MK.EXE. Эту программу и её описание можно найти на сайте производителя ЭКВМ: http://mk.semico.ru/progmk.htm

После установки Каллисто, вероятно, потребуется также передать в МК-161 редактор и другие программы, написанные на Каллисто.

Для установки Каллисто на калькулятор:

- Передайте на калькулятор файл `Callisto.mkp` — он занимает 99 страниц памяти программ «Электроники МК-161», начиная со страницы №0.
  - Исполняемый файл занимает 9883 байт. Контрольная сумма ADD 761001. Контрольная сумма XOR 191.
- Сохраните этот файл на электронный диск калькулятора. Можно использовать имя по вашему выбору, латинским шрифтом или кириллицей.
- Запустите программу обычным образом: `В/О С/П`.

Чтобы передать на калькулятор программы, написанные на Каллисто:

- Передайте файлы с расширением MKT из каталога `Blocks-source` на калькулятор по одному файлу.
- Сохраните каждый передаваемый файл на электронный диск калькулятора в каталог Каллисто1 как файл типа Т (текст). Используйте имена файлов, с которыми эти файлы поставляются, без расширения. Например, файл `Blocks-source/B011.mkt` нужно сохранить как B011 (латинская B и три цифры 011).

Важно: файлы в каталоге `Blocks-source` содержат полезные инструменты и примеры. В частности, файл `Blocks-source/B011.mkt` содержит исходный код *строкового редактора EDIT* — стандартное средство Каллисто для написания программ. Этот (или подобный) редактор необходим для полноценного программирования на Каллисто.

Сразу после установки Каллисто и передачи блоков (файлов MKT) рекомендуется откомпилировать редактор командой `11 LOAD` (компиляция может занять порядка 15 минут) и сохранить редактор на диске командой `SAVE" EDIT`. (Подробнее о редакторе см. главу 8). Это же относится ко всем программам, которые вы намерены часто использовать на своей «Электронике». Загрузка скомпилированных программ происходит значительно быстрее, чем их интерпретация из блоков! Можно откомпилировать сразу несколько программ, если на это хватит памяти, и сохранить получившийся комплект под одним именем.

Чтобы не тратить время на компиляцию, можно загрузить на калькулятор откомпилированные версии тех же программ из каталога `Blocks-compiled`. Каждая из пяти откомпили-

рованных программ состоит из двух файлов с расширениями MKB и MKD, которые надо записать на электронный диск ЭКВМ в виде файлов типа B и D под одинаковым именем, под которым они загружаются из Каллисто словом `LOAD"`

Однако если вы измените `Callisto.mkl` и самостоятельно скомпилируете новую, отличную от дистрибутива, версию `Callisto.mkp`, готовые скомпилированные файлы работать не будут. Потребуется заново компилировать программы получившейся версией Каллисто. Номер версии Каллисто при правке исходного кода транслятора также имеет смысл изменить.

3.2 Установка Каллисто на эмулятор калькулятора

Чтобы опробовать Каллисто без физического калькулятора, вы можете использовать эмулятор eMkAtic.

Установите на эмуляторе «накладку» клавиатуры, находящуюся в файле `kbd\CallistoKbdEmu.png` в архиве поставки.

При помощи меню эмулятора можно передать файлы на диск эмулируемого калькулятора. Следуйте той же процедуре, что и при установке на реальный калькулятор.

4 Клавиатура калькулятора для Каллисто

Каллисто переопределяет все клавиши МК-161. Новые значения клавиш вы можете увидеть на Рис. 1 (страница 2) этого руководства.

Распечатайте Рис.1 на листе А4 и прорежьте дырочки под клавиши — работать с накладкой намного удобней, чем угадывать клавишу или вычислять её по рисунку.

Для вырезания дырок под клавиши ВКЛ и подсветки лучше всего использовать пробойник на 6 мм. Для остальных клавиш можно использовать пробойник на 4 мм — или даже на 3 мм, если умеете пробивать дырки очень точно. При отсутствии инструмента можно использовать обычные маникюрные ножницы, но потребуются терпение и аккуратность.

При старте Каллисто клавиатура находится в цифровом режиме, максимально совпадающем с заводскими надписями на клавишах ЭКВМ. Для ввода английских букв нажмите чёрную клавишу ЛАТ (Р-ГРД-Г). Буквы расположены на клавишах «Электроники-161» в алфавитном порядке слева направо и сверху вниз. Синяя клавиша NUM (Р) возвращает клавиатуру в цифровой режим.

Чтобы набирать русские буквы, перед ЛАТ (Р-ГРД-Г) нажмите оранжевую клавишу F. В русском языке 33 буквы, поэтому на русской клавиатуре клавиши ESC (ВЫХОД), ВВОД и четыре «стрелки» работают на ввод русских букв Д, Е, А, Б, В и Г. Когда вам потребуется ввести ESC или перевести строку, вернитесь на цифровую клавиатуру нажатием клавиши NUM (Р).

Для ввода заглавной буквы нажмите перед ней оранжевую клавишу F. Изменить регистр вводимых букв позволяет комбинация F NUM (CAPS или РЕГИСТР). В Каллисто регистр букв роли не играет, заглавные или строчные буквы в большинстве случаев взаимозаменяемы.

Каллисто нельзя остановить клавишей С/П. Для выхода из Каллисто служит слово BYE. Также можно выключить МК-161, отключить его от зарядного устройства и включить обратно, удерживая нажатой клавишу С (С/П).

Красная клавиша BS (Cx) удаляет последний введённый символ, комбинация F BS удаляет всю введённую строку и позволяет начать ввод заново. Клавиша ВВОД (недоступна на русской клавиатуре) передаёт введённую строку Каллисто — на исполнение или компиляцию. Клавиша ВП вводит пробел.

Для ввода порядка вместо ВП используется английская буква E, например 1E50 вместо 1 ВП 50. Чтобы её набрать, переключитесь на английскую клавиатуру, а потом обратно.

5 Введение в язык Форт, диалект Каллисто

Форт — язык, созданный до многих современных языков и не похожий на них. С другой стороны, некоторые концепции Форта очень напоминают привычный способ программирования микрокалькулятора.

Мы опишем основы языка Форт в диалекте Каллисто очень кратко. Чтобы освоить полноценное программирование на Форте, мы рекомендуем обратиться к литературе, например:

Leo Brodie “Starting Forth”. https://www.forth.com/starting-forth/ . Существует перевод старой версии на русский язык - http://www.nncron.ru/book/sf/

Баранов С.Н., Ноздрунов Н.Р. «Язык Форт и его реализации» — Л.: Машиностроение. Ленингр. отд-ние, 1988.. Эту книгу и другие книги на русском языке можно найти по ссылкам со страницы http://www.forth.org.ru/forth/lit.html

Семёнов Ю.А. «Программирование на языке Форт» — М.: Радио и связь, 1991. Эту книгу можно найти в Интернете, например, здесь: http://publ.lib.ru/ARCHIVES/S/SEMENOV_Yuriy_Alekseevich/_Semenov_Yu.A..html

Чтобы применять информацию из книг о Форте для работы с Каллисто, прочтите главу «Главные отличия Каллисто от стандартного Форта» данного руководства.

Подробную информацию о словах (включая подробные таблицы слов) и других особенностях Каллисто можно найти на вики http://pmk.the-hacker.ru

5.1 Программирование на Форте: слова

Основной элемент языка Форт — *слово.* Слово — это любое, абсолютно любое, поименованное действие. Более того, для Форта данные (константы, переменные и т. п.) - в конечном счёте тоже действия, и потому — тоже слова.

В Форте нет никаких параметров, определяемых скобками блоков (как () в LISP или {} в Си) и других особенностей, привычных современному программисту. Программа на Форте — это последовательность слов. А управление машиной (в данном случае ЭКВМ) в Форт-среде — это ввод последовательности слов. Между словами должен быть пробел (клавиша `ВП`) или перевод строки (клавиша `ВВОД`).

Собственно говоря, программирование и управление машиной не различаются. Форт-среда может принимать последовательность слов с клавиатуры, а может — из блоков диска, в которых записано приложение. Исполняются они абсолютно одинаково.

Форт-среда — одновременно и интерпретатор, и компилятор. Если введено (с клавиатуры или из блока памяти — это неважно) существующее слово, она исполняет его сразу — это называется *состояние исполнения*.

Также вы можете в любой момент определить новое слово. Во время определения оно не исполняется — вместо этого Каллисто записывает его в *словарь* в некоем внутреннем формате, который позволяет максимально быстро исполнить его позже. Это — *состояние компиляции*. Когда определение нового слова завершено, система возвращается в состояние исполнения.

Считается, что идеальная программа на Форте определяет каждое нужное действие как одно слово — на том уровне, на каком это требуется программисту. Так, Лео Броуди приводит пример алгоритма работы стиральной машины:

```
: WASHER  WASH SPIN RINSE SPIN ;
```

Здесь каждое действие стиральной машины определено одним словом — стирка (wash), отжим (spin), полоскание (rinse). Заметим, что отжим использован два раза. Предполагается, что раньше программист уже описал на Форте все процессы управления машиной в каждом из этих режимов, определив слова `WASH`, `SPIN`, `RINSE`.

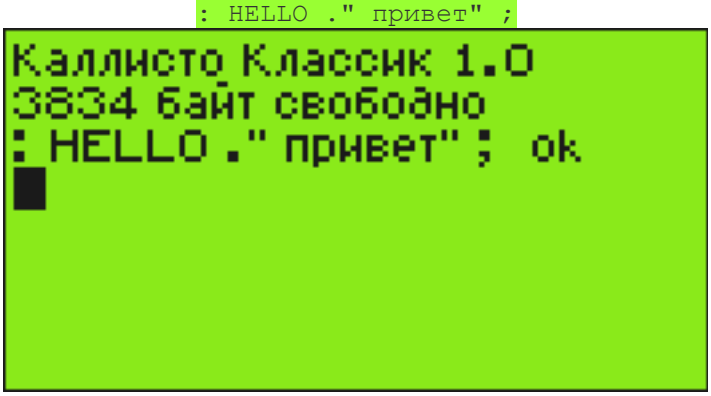
А сама строка в этом примере определяет новое слово WASHER как последовательность из этих слов. И теперь, чтобы запустить стирку, достаточно ввести с клавиатуры одно-единственное слово:

WASHER

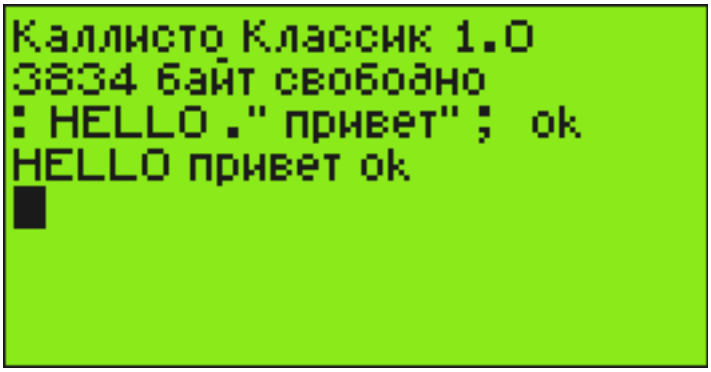
Итак, для определения новых слов используются литеры : и ; . Точнее, слова : и ; — как мы уже упомянули, в Форте всегда используются слова. Поскольку это слова, перед ними и после них требуется пробел (или перевод строки). После слова : указывается имя нового слова, а затем слова, которые в него входят, вплоть до слова ; . Слово ; завершает создание нового слова.

Определим простое слово, работающее на Каллисто:

```
: HELLO ." привет" ;
```



Теперь, если ввести HELLO и нажать клавишу ВВОД, система ответит «привет».



Здесь использовано слово ." - оно выводит на экран всё, что следует за ним, вплоть до литеры " . *Поскольку это именно слово, после него и перед выводимой информацией обязательно должен быть пробел*. Правильно ." привет", но ни в коем случае не ."привет" — это важное отличие Форта от других языков.

Заглавные и строчные буквы в именах слов не различаются.

Когда вы управляете системой с клавиатуры, вы можете ввести несколько слов одновременно, разделяя их пробелами (клавиша ВП). Чтобы система выполнила то, что вы ввели, нажмите клавишу ВВОД. Если вы допустили ошибку, система выводит слово, которое вызвало ошибку, и код ошибки. (При этом пропадает вся информация из стека, который описан ниже). Если же все слова выполнены удачно, система показывает сообщение ok и снова готова к вводу слов.

Осторожно: Некоторые ошибки, например, деление на ноль или вычисление квадратного корня из отрицательного числа, останавливают Каллисто. Чтобы начать заново, нажмите В/О С/П — при этом Каллисто запустится заново, не сохранённый на диск результат работы пропадёт.

Важно: в этой главе сообщается о многих категориях слов Каллисто и описаны некоторые из слов. Полный список слов в каждой категории можно увидеть на странице «Слова Каллисто по категориям» вики http://pmk.the-hacker.ru .

5.2 Работа с числами: стек и математические операции

В языке Форт есть только слова. Нет параметров, нет формул и т. п. Написать на нём обычное арифметическое выражение, например «2+2», невозможно.

Вместо этого для работы с числами используется *стек*. Это список чисел, организованный по принципу LIFO – last in, first out. «Последним вошёл — первым вышел». Стек можно сравнить со стопкой тарелок — в каждый момент вы можете взять только верхнюю тарелку, затем ту, что под ней и т. д., а можете и добавить новую тарелку, и тогда она станет верхней.

Чтобы «положить» число в стек — добавить его и сделать верхним элементом — достаточно просто ввести его. Любое число, целое или дробное, Каллисто воспринимает как слово, означающее «положить данное значение в стек».

Арифметические операции проводятся с числами в стеке. Например, слово `+` извлекает из стека два числа, складывает их и кладёт результат в стек.

Чтобы увидеть результат, используйте слово `.` (точка). Оно берёт число из стека и выводит на экран.

Теперь мы можем сложить два и два:

```
2 2 + .
```

После нажатия клавиши `ВВОД` система выведет результат: `4`.

Важно, что слово `.` забирает число из стека. Если вы хотите вывести число, а затем продолжить работать с ним, нужно сначала продублировать его на стеке. Для этого служит слово `↑` (в обычном Форте оно называется DUP; в любом учебнике по Форту это слово очень активно применяется). После выполнения этого слова верхнее число в стеке повторится дважды. Теперь, если вывести его словом `.` , на вершине стека снова останется одна копия этого же числа.

Существуют также слова, позволяющие манипулировать значениями на стеке, например, `↔` меняет местами два верхних значения (в обычном Форте это слово называется SWAP). При помощи слов для манипуляции стеком можно выполнять более сложные вычисления. Примеры вы можете найти в литературе по языку Форт.

Способ вычислений в Форте похож на применяемый на калькуляторе в его обычном режиме: `2 В↑ 2 +` . Но в отличие от обычного режима калькулятора, где стек состоит лишь из нескольких регистров, в Каллисто стек бесконечный, точнее, может хранить многие сотни значений. При правильно составленной программе переполнение стека маловероятно.

Все математические операции калькулятора также доступны. Но для них нет специальных клавиш; не используются и клавиши-модификаторы `F` и `K`. Так, например, вместо `Fsin` можно ввести слово `SIN`, а вместо `K{x}` слово `{X}` (дробная часть числа).

Для переключения градусной меры угла служат слова `DEGREES RADIANS`. Они влияют на исполнение всех тригонометрических функций.

Слово `y^x` возводит число в степень. Например, `4 3 y^x` - возвести 4 в куб. (Заметим, что на вершине стека должен быть показатель степени — в этом примере 3).

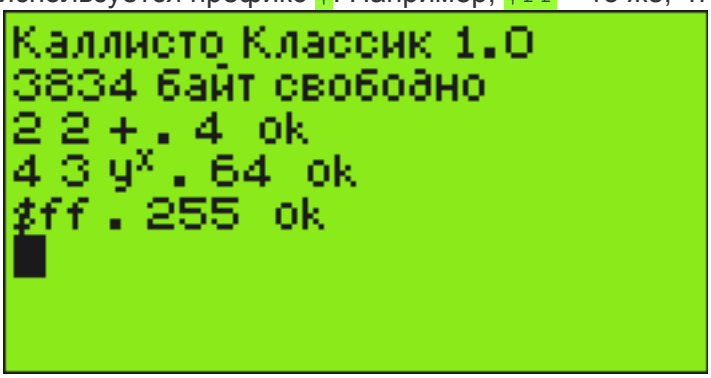
Есть и побитовые операции, работающие с целой частью двух чисел так, как если бы это были 32-битные целые числа: слова `AND`, `OR`, `XOR`. А слово `NOT` снимает число со стека, инвертирует (логическое отрицание) все его биты (опять же — целой части как 32-битного числа) и записывает результат на стек.

Также вы можете изменить систему счисления, в которой вы вводите числа и система выводит их на индикатор (экран). Слово `HEX` устанавливает шестнадцатиричную систему счисления, а `DECIMAL` — десятичную. Можно также установить произвольную систему счисления, например, для восьмеричной системы:

```
8 BASE !
```

Эта конструкция присваивает стандартной переменной `BASE` значение 8; работа с переменными описана ниже.

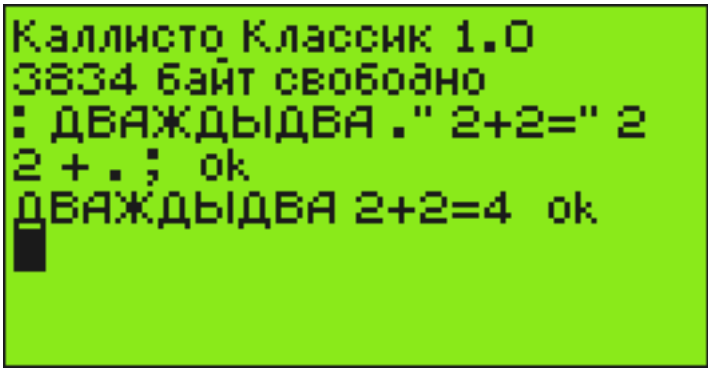
Можно также ввести отдельное шестнадцатиричное число, не меняя значение переменной `BASE`. Для этого используется префикс `$`. Например, `$FF` – то же, что `255`.



Конечно, любые арифметические операции можно включить и в определение новых слов. Например:

```
: ДВАЖДЫДВА ." 2+2=" 2 2 + . ;
```

Теперь, введя слово `ДВАЖДЫДВА`, вы увидите ответ `2+2=4`.



Внимание: вы можете работать с числами с десятичной запятой, так же, как в обычном режиме ЭКВМ. При этом для разделения целой и дробной части используется именно *запятая*, как в калькуляторе, а не точка, как в иностранных языках программирования.

5.3 Стековая нотация слов

Стандартный способ документирования слова в Форте вообще и в Каллисто в частности — *стековая нотация*. Такое описание показывает, какие данные слово берёт со стека и какие — кладёт на стек.

Для стековой нотации обычно используют скобки. Сначала пишут данные, которые слово берёт со стека, затем литеры `--` и, наконец, данные, которые слово кладёт на стек:

```
( данные со стека -- данные на стек )
```

Например, слово `1+`, которое снимает число со стека, прибавляет к нему единицу и кладёт результат на стек, описывается так:

```
1+ ( x -- x+1 )
```

Текст, написанный в скобках, в Форте считается комментарием и не выполняется. Поэтому при определении слова можно сразу же ввести его стековую нотацию. Такое определе-

ние легче читать. Например, определим слово `3+`, которое снимает число со стека, прибавляет к нему 3 и кладёт результат на стек

```
: 3+ ( x -- x+3 ) 3 + ;
```

Это слово кладёт на стек 3, а затем вызывает слово `+`. Слово `+` снимает с вершины стека два числа, первым из них оказывается 3, а вторым — то, которое лежало на стеке до исполнения нового слова `3+`. Результат сложения оказывается на стеке.

Внимание: на самом деле комментарий в Форте реализуется словом `(` . Это слово игнорирует всю информацию до литеры `)` . Поскольку `(` это именно слово, *после открывающей скобки обязательно должен быть пробел*. Вложенные комментарии не поддерживаются — первая же литера `)` закрывает комментарий.

Если слово не берёт ничего со стека, то до литер `--` в скобках просто ничего не пишется (оставляют пробел). А если слово ничего не кладёт на стек, ничего не пишется после литер `--` . Таким образом, слово `HELLO`, которое мы определили выше, можно описать так:

```
HELLO ( -- )
```

Когда слово берёт со стека два или более числа, число, находящееся на верхушке стека, записывают на диаграмме *правее* остальных. Именно так эти числа надо вводить перед вызовом слова. Если ввести, например, слова `12` `14` , то на верхушке стека окажется 14 и уже под ним 12.

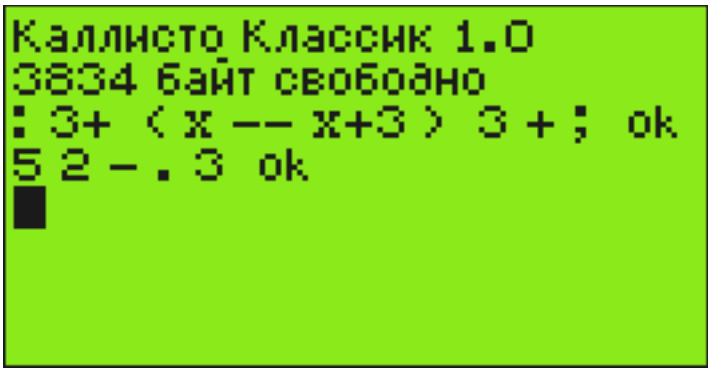
Например, встроенное слово `-` (вычитание) описывается так:

```
- ( y x -- y-x )
```

При этом на верхушке стека должно быть x (вычитаемое) и под ним y (уменьшаемое). Вычислим 5-2:

```
5 2 - .
```

Каллисто выведет на экран `3`.



5.4 Работа с данными: переменные, величины, константы

В первых версиях языка Форт работать с числами можно было исключительно при помощи стека. Если предполагаются сложные последовательные операции со многими числами, эта работа превращается в головоломку — постоянно нужно рассчитывать, какое число на вершине стека в данный момент в программе. Чак Мур, создатель Форта, считал, что Форт в качестве упражнения для мозга сильно превосходит карточные игры, кроссворды и судоку.

В реальном программировании, однако, чрезмерная головоломность не всегда хороша. К счастью, в более современных версиях языка Форт, и в частности в Каллисто, числа можно хранить и иным образом. Чтобы с ними что-то делать, их всё равно приходится переводить в стек — но нет необходимости думать, какое место занимает в стеке каждое число в каждый момент.

Для хранения целых чисел используются *переменные* — достаточно старая и уважаемая конструкция в Форте. Переменная — это особое слово. Чтобы создать слово и определить его как переменную, используется слово `VARIABLE`, например:

```
VARIABLE MY_VAR
```

Чтобы присвоить переменной числовое значение, используется слово `!` , например:

```
3 MY_VAR !
```

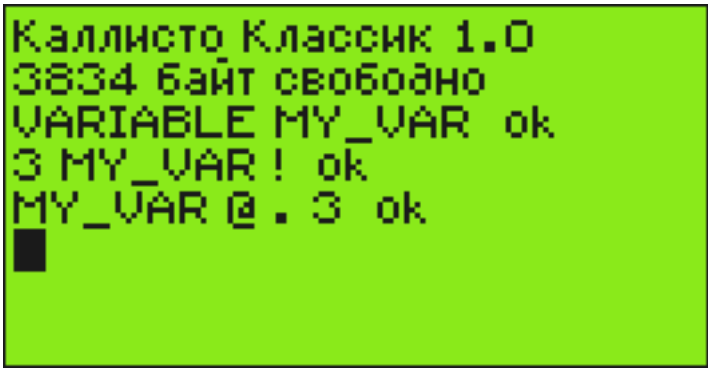
В любой момент значение переменной можно скопировать на верхушку стека при помощи слова `@`.

```
MY_VAR @
```

После этого его можно обрабатывать обычным образом. Например, показать на экране:

```
MY_VAR @ .
```

Благодаря слову `.` число будет потеряно из стека, но оно всё равно сохранится в переменной.



Для дробных чисел, привычных для калькулятора, используется другой вид переменной — *величина*. Это более новая конструкция для Форта и потому работа с ней отличается от работы с обычной переменной.

Чтобы создать слово и определить его как величину, используется слово `VALUE`, например:

```
0 VALUE MY_VALUE
```

Важно, что слово `VALUE`, в отличие от `VARIABLE`, берёт первоначальное значение переменной со стека. Поэтому значение нужно положить на стек перед словом `VALUE`, в нашем примере используется значение 0.

Чтобы присвоить величине новое числовое значение, используется слово `TO`, например:

```
2,71828 TO MY_VALUE
```

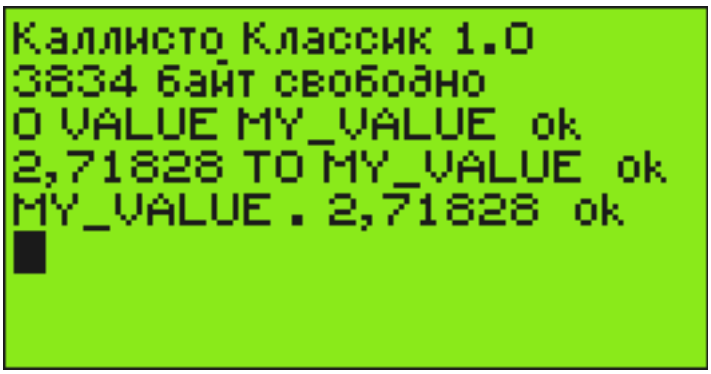
Для того, чтобы скопировать значение величины на верхушку стека, нужно просто ввести имя переменной как слово:

```
MY_VALUE
```

После этого его можно обрабатывать обычным образом. Например, показать на экране:

```
MY_VALUE .
```

Благодаря слову `.` число будет потеряно из стека, но оно всё равно сохранится в величине.

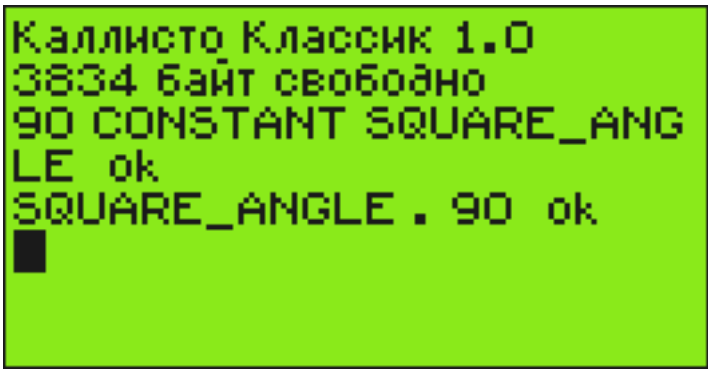


Слово можно также определить как константу – то есть постоянное целое числовое значение, которое не будет меняться. Для этого значение константы нужно положить на стек, а затем использовать слово CONSTANT, например:

```
90 CONSTANT SQUARE_ANGLE
```

Чтобы положить на стек значение константы, достаточно ввести название константы как слово (как и в случае величины). Например, можно показать содержимое константы на экране:

```
SQUARE_ANGLE .
```



5.5 Условный оператор

В программе нередко требуется выполнить различные действия в зависимости от разных условий — например, только если значение А меньше значения Б. В большинстве языков программирования для этого используется та или иная форма оператора IF. Каллисто — не исключение.

Внимание: слово `IF` разрешено только в рамках определения нового слова (между словами `:` и `;`). Если ввести слово `IF` в режиме интерпретации, то есть вне определения нового слова, среда Каллисто выдаст сообщение об ошибке.

В отличие от многих других языков, слово `IF` в Форте не включает в себя саму проверку условия (например, сравнение А и Б).

Вместо этого в программе должно использоваться отдельное слово проверки. Оно оставляет на вершине стека результат проверки, называемый *флаг* — 0 (ложь) или -1 (истина). Например, слово проверки `< ( y x -- результат )` кладёт на стек -1, если y меньше x, и 0 в противном случае.

Число -1 используется потому, что в виде 32-битного целого числа оно представлено всеми 32 битами единиц. Поэтому такие значения позволяют использовать побитовые логические операции как обычные логические операции. Например, если на верхушке стека лежит 0 и вызывается слово `NOT`, там окажется -1. А из -1, в свою очередь, слово `NOT` сделает 0.

Само слово `IF` снимает флаг с вершины стека и, в зависимости от его значения, выполняет или не выполняет действия. Это слово применяется в двух вариантах:

«`IF` *действия* `THEN`» выполняет все действия (последовательность слов) между `IF` и `THEN`, если флаг на вершине стека — не 0. А если 0 — ничего не выполняет. Заметим, в отличие от других языков, слово `THEN` *завершает* конструкцию.

«`IF` *действия1* `ELSE` *действия2* `THEN`» выполняет *действия1*, если флаг — не 0. А если 0 — выполняет *действия2*.

Например, сравним значения А и Б. (Сами А и Б должны быть определены как `VALUE`).

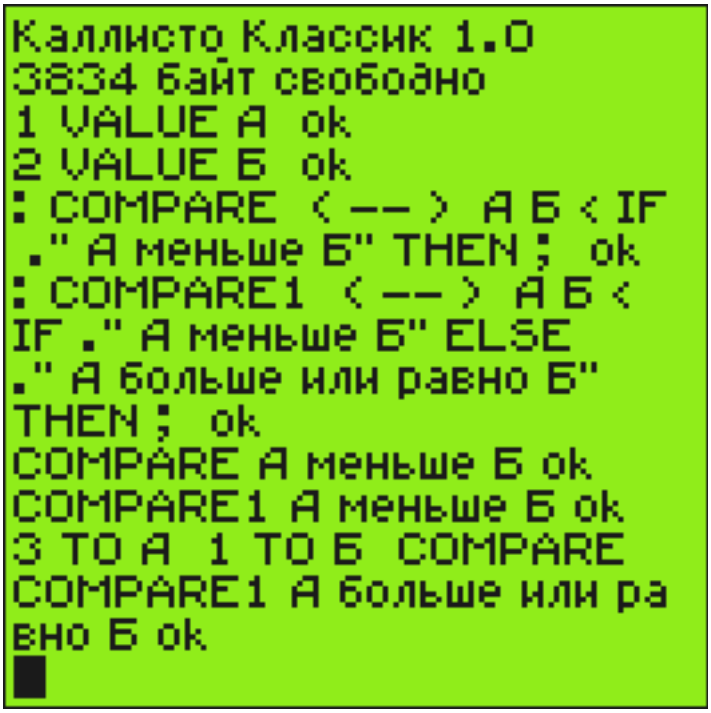
```
1 VALUE А
2 VALUE Б
: COMPARE ( -- ) А Б < IF ." А меньше Б" THEN ;
```

```
: COMPARE1 ( -- ) А Б < IF ." А меньше Б" ELSE ." А больше или равно Б" THEN ;
```

Теперь слово COMPARE выдаст «А меньше Б», только если А действительно меньше Б. А слово COMPARE1 сделает то же самое, но если А не меньше Б, выдаст другое сообщение.

В данном примере А меньше Б, но вы можете присвоить А и Б другие значения, а затем слова выполнить слова COMPARE и COMPARE1, например:

```
3 TO А 1 TO Б COMPARE COMPARE1
```



5.6 Цикл FOR со счётчиком

Каллисто поддерживает несколько видов циклов, позволяющих выполнить одни и те же действия несколько раз. Самый распространённый цикл, «со счётчиком», реализован словом FOR. Перед выполнением этого слова на верхушке стека должно быть целое число, определяющее, сколько раз цикл будет исполнен. (Если на верхушке стека 0 или отрицательное число, цикл не исполняется вовсе).

Внимание: слово FOR разрешено только в рамках определения нового слова (между словами : и ;). Если ввести слово FOR в режиме интерпретации, то есть вне определения нового слова, среда Каллисто выдаст сообщение об ошибке.

FOR исполняет в цикле все действия (называемые *тело цикла*) до слова NEXT. Можно также получить доступ к значению счётчика (сколько раз осталось выполниться циклу) — слово I кладёт его на стек. Например:

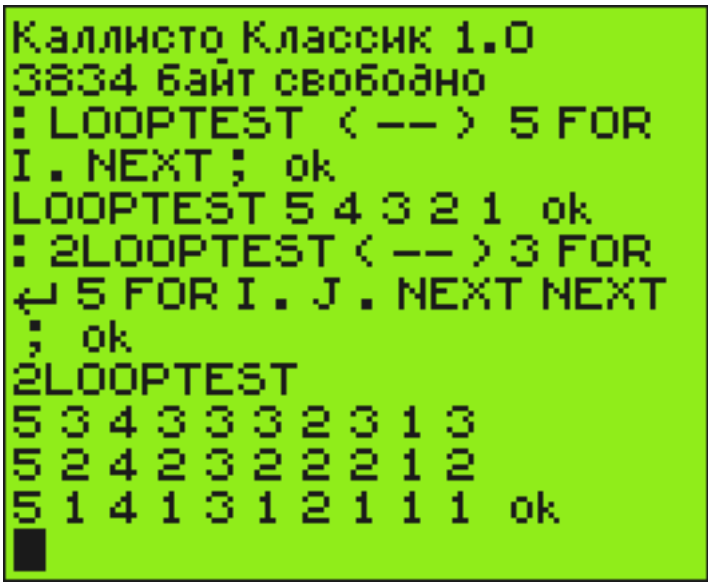
```
: LOOPTEST ( -- )  5 FOR I . NEXT ;
```

Слово LOOPTEST теперь выведет значение счётчика каждое выполнение цикла. (Счётчик начинается с заданного значения и уменьшается на 1 каждый раз, поэтому вывод в данном случае будет 5 4 3 2 1).

Внутри цикла FOR можно выполнить ещё один цикл FOR. Такой цикл называется *вложенным*. В этом случае, чтобы получить на стек счётчик внешнего цикла (того, в который вложен текущий цикл), используется слово J. В качестве примера вы можете определить слово:

```
: 2LOOPTEST ( -- )  3 FOR ↵  5 FOR I . J . NEXT NEXT ;
```

При каждом исполнении внутреннего цикла будет выведен и внутренний, и внешний счётчик. Слово ↵ начинает вывод с новой строки.



Вложить во вложенный цикл ещё один цикл и т. д. тоже можно — количество вложений почти не ограничено. Счётчик внутреннего цикла доступен как `I`, того, в который он вложен — `J`, следующий уровень — `K`. Если циклы вложены друг в друга четыре раза и более, счётчики внешних циклов недоступны.

Слов `LEAVE` немедленно выходит из цикла FOR (аналог оператора break во многих языках). Далее выполняется слово, следующее за словом `NEXT`. Если несколько циклов вложено друг в друга, `LEAVE` выходит только из внутреннего цикла.

В Каллисто, чтобы выйти из двух и более циклов FOR сразу, используется слово `UNLOOP`. Так, `UNLOOP LEAVE` выходит сразу из двух вложенных циклов, `UNLOOP UNLOOP LEAVE` – из трёх и т. п.

Можно также выйти не только из цикла, но и из всего слова, в котором цикл определён. Для этого используется сочетание слов `UNLOOP EXIT`.

Внимание: ошибка в количестве слов `UNLOOP` может привести среду Каллисто в непредсказуемое состояние. Тот же результат возможен, если в цикле FOR использовать слово `EXIT`, не исполнив перед ним слово `UNLOOP`.

Заметим, что цикл FOR в Каллисто является аналогом циклов ЭКВМ со счётчиком `FL0 FL1 FL2 FL3`.

5.7 Циклы BEGIN

Иногда заранее неизвестно, сколько раз нужно повторить какое-то действие. Приходится сначала его выполнить, а потом проверить — закончена работа или надо повторить ещё раз. Такой цикл «с постусловием» существует во многих языках, например, в Паскале и Обероне он называется repeat...until.

В Каллисто это цикл BEGIN-UNTIL. Он определяется так:

```
BEGIN действия UNTIL
```

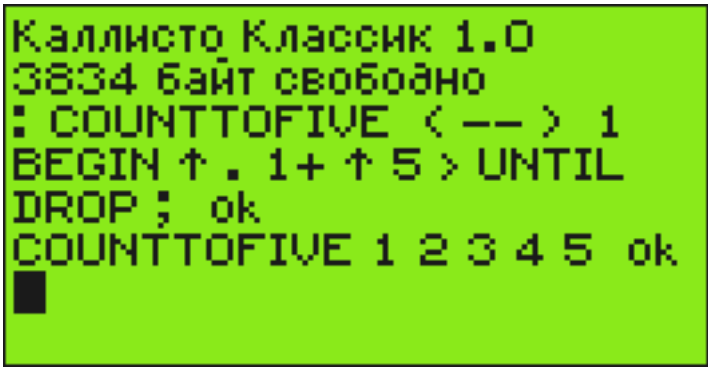
Действия — это последовательность слов (тело цикла). После исполнения действий, перед словом `UNTIL`, на стеке должно быть значение-флаг (как для слова `IF`). Слово `UNTIL` снимает флаг со стека, и если это 0, цикл выполняется ещё раз, а если не ноль — цикл завершается.

В качестве примера мы реализуем при помощи BEGIN-UNTIL цикл со счётчиком, который будет увеличиваться, а не уменьшаться.

```
: COUNTTOFIVE ( -- )  1 BEGIN ↑ . 1+ ↑ 5 > UNTIL DROP ;
```

Слово `COUNTTOFIVE` кладёт на стек единицу, а затем начинает цикл. В цикле оно сначала дублирует на стеке и выводит значение счётчика. (Дублирование нужно потому, что при выводе число на вершине стека теряется). Затем значение счётчика на стеке увеличивается на 1 (слово `1+`) и снова дублируется, теперь уже для сравнения с 5. Если увеличенное значение оказалось больше 5, цикл заканчивается. Слово `DROP` убирает счётчик со стека, чтобы состояние стека осталось таким же, каким было до выполнения слова `COUNTTOFIVE`.

Таким образом, слово `COUNTTOFIVE` выведет на экран `1 2 3 4 5`.



Во многих языках есть также цикл, проверяющий условие *перед* выполнением тела цикла. Обычно он называется while. Его место в Каллисто занимает цикл BEGIN-WHILE-REPEAT, который позволяет подготовить условие, проверить его, а затем выполнить тело цикла:

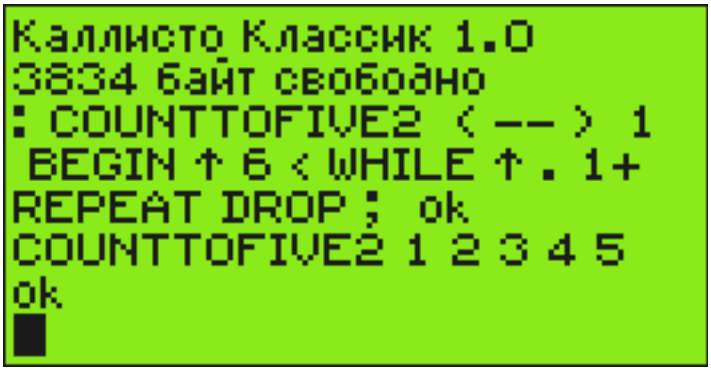
`BEGIN` *`действия1`* `WHILE` *`действия2`* `REPEAT`

Слова «*действия1*» выполняются в начале. Слово `WHILE` снимает флаг со стека. Если флаг равен 0, цикл немедленно завершается (исполняется следующее слово после `REPEAT`). Если же флаг не равен 0, выполняются «*действия2*», затем снова «*действия1*» и снова проверка флага.

Внимание: значение флага в BEGIN-WHILE-REPEAT противоположно значению в BEGIN-UNTIL. Флаг 0 продолжает цикл BEGIN-UNTIL, но завершает BEGIN-WHILE-REPEAT.

Например, тот же счёт до пяти можно реализовать и так:

```
: COUNTTOFIVE2 ( -- )   1 BEGIN ↑  6 < WHILE ↑ . 1+ REPEAT DROP ;
```



Если в цикле выполнено слово `EXIT`, происходит выход из слова, внутри которого определён цикл. При этом выполнение данного цикла немедленно прекращается, как в операторе return многих других языков. (В циклах BEGIN, в отличие от цикла FOR, не нужно использовать слово `UNLOOP`). Обычно слово `EXIT` используется в рамках конструкции `IF`, чтобы прервать цикл в каком-либо особом случае.

Более того — существует и «бесконечный» цикл BEGIN-AGAIN. Все слова между `BEGIN` и `AGAIN` будут повторяться бесконечно, и для выхода используется только слово `EXIT`.

Внимание: Циклы BEGIN разрешены только в рамках определения нового слова (между словами `:` и `;`). Если ввести слово `BEGIN` в режиме интерпретации, то есть вне определения нового слова, среда Каллисто выдаст сообщение об ошибке.

5.8 Выделение памяти, работа со строками и массивами

Стек, переменные и величины позволяют работать с числами. Но что, если нужно обрабатывать строки? Или массивы из чисел?

В языке Форт нет таких типов данных. Зато в нём есть способ выделить некоторое количество памяти, которое вы затем можете использовать по своему усмотрению — обычно, как раз, для строк или массивов.

Выделить память можно в области, специально выделенной для слов. Эта область называется *словарь*. Часть словаря в любой конкретный момент является занятой, а часть — свободной. Среда Каллисто хранит адрес первого свободного байта.

Для выделения памяти используются конструкции из слов `CREATE` и `ALLOT`. Например, вот так можно выделить 10 байт памяти:

```
CREATE 10BYTES 10 ALLOT
```

Слово `10BYTES` теперь указывает на начало выделенного участка.

Как работает эта конструкция? Слово `CREATE` создает новое слово, которое именует некий байт памяти, а именно — первый свободный байт словаря (на данный момент).

Имя нового слова берётся из «буфера ввода», а содержимым буфера считается то, что введено с клавиатуры непосредственно после слова `CREATE`.

Важно, однако, что сам этот байт считается ещё не занятым. Поэтому нельзя использовать вновь созданное слово сразу после слова `CREATE`. Ведь оно указывает на байт, который ещё не считается занятым — и потому может быть занят чем-то другим, а точнее, заголовком следующего созданного слова. Поэтому после слова `CREATE`, как правило, требуется слово `ALLOT`.

Слово `ALLOT` берёт со стека значение (в данном случае 10) и отмечает, что данное количество байт в словаре, начиная с текущего верхнего байта, зарезервировано. А точнее — просто изменяет указатель на первый свободный байт так, что новый свободный байт оказывается выше старого (в нашем примере — на 10 байт). И потому следующее выделение памяти начнётся уже с нового свободного байта и продолжится ещё выше, а зарезервированная область останется нетронутой. Её можно спокойно использовать для любых данных.

Указатель на первый свободный байт словаря можно положить на стек словом `HERE`. После слов `CREATE 10BYTES` оказывается, что `10BYTES` и `HERE` «указывают» на один и тот же байт — то есть слово `10BYTES` положит на стек тот же адрес, что и слово `HERE`. Но `10 ALLOT` увеличивает значение `HERE` на 10.

Наше новое слово `10BYTES` уже можно использовать как массив. Обратите внимание, что значения в этом массиве — байты, т.е. целые числа от 0 до 255.

Чтобы записать байт в память, нужно положить на стек значение, затем адрес байта в памяти и исполнить слово `C!` . Для нулевого элемента массива (10BYTES[0]) всё просто — слово `10BYTES` кладёт на стек адрес начала массива, т.е. этого элемента. Запишем в него число 25:

```
25 10BYTES C!
```

А чтобы записать значение в другой элемент, нужно прибавить к адресу нужное количество байт. Например, запишем 37 в 10BYTES[3]:

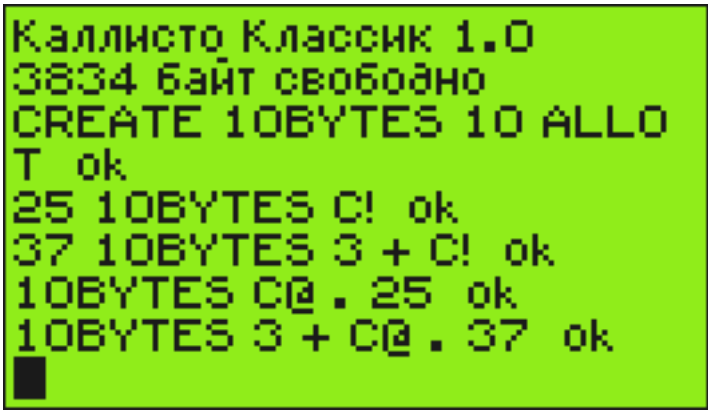
```
37 10BYTES 3 + C!
```

Чтобы положить на стек байт из памяти по некоему адресу, следует положить этот адрес на стек и вызвать слово `C@` . Выведем содержимое 10BYTES[0]:

```
10BYTES C@ .
```

А теперь выведем содержимое 10BYTES[3]:

```
10BYTES 3 + C@ .
```



Существуют также слова `@` и `!` , которые считывают и записывают двухбайтовые значения по указанному адресу. С ними нужно обращаться осторожно — к указателям тоже всегда нужно прибавлять 2, 4, 6 и т. п. На «Электронике МК-161» по указанному адресу находится старший байт, а следом за ним — младший байт.

Слово `@` считывает двухбайтовое значение со знаком. Для чтения беззнакового двухбайтового значения (от 0 до 65535) служит уникальное каллистянское слово `U@` .

Но иногда требуется сохранить в массив совсем не байты и не двухбайтовые значения, а полноценные десятичные числа (с плавающей запятой). Для этого используется отдельный *десятичный словарь*. В нём тоже можно поименовать словом ячейку памяти, но по ряду причин для этого вместо слова `CREATE` используется сочетание `ДHERE CONSTANT`. А вместо `ALLOT` для десятичного словаря используется `ДALLOT` (`Д` — русская литера, остальные — латинские). Создадим простейший массив из десяти чисел:

```
ДHERE CONSTANT 10VALUES 10 ДALLOT
```

Слово `10VALUES` теперь указывает на выделенную области из десяти ячеек в десятичном словаре. (Кстати, на верхнюю ячейку десятичного словаря указывает слово `ДHERE`).

Присвоим значение 10VALUES[0]=2,718:

```
2,718 10VALUES C!
```

Присвоим значение 10VALUES[3]=3,141:

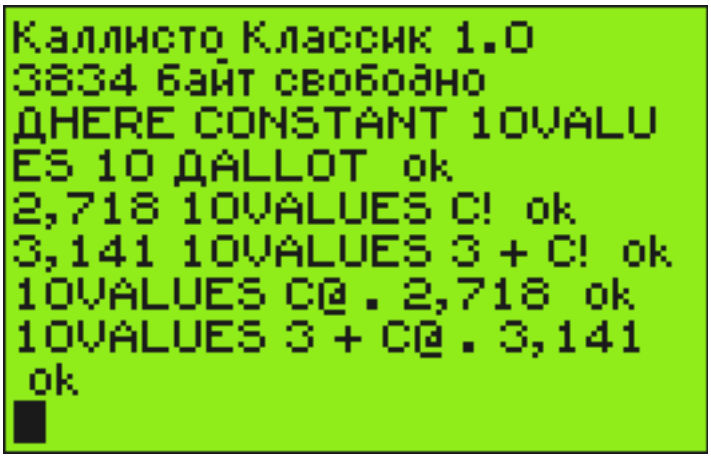
```
3,141 10VALUES 3 + C!
```

Выведем содержимое 10VALUES[0]:

```
10VALUES C@ .
```

Выведем содержимое 10VALUES[3]:

```
10VALUES 3 + C@ .
```



5.9 Слова для порождения новых слов

Язык Форт — чрезвычайно гибкий, расширяемый инструмент. Вы уже знаете, что, в отличие от привычных языков программирования, главный способ расширения Форта — создание слов, которые можно применять точно так же, как встроенные слова среды. Также вам знакомо применение слов `CREATE` и `ALLOT` для создания слов, указывающих на данные в памяти.

Существует также дополнительный механизм для создания слов, работающих с данными в памяти более сложным образом. Вы можете создать *порождающее слово*, которое в свою очередь позволяет создавать другие, *порождённые* слова.

Каждое порождённое слово «привязано» к своему экземпляру данных в памяти. Но вот обработка этих данных будет производиться одним и тем же способом, заданным в порождающем слове.

Этот уникальный механизм можно сравнить с определением новых типов данных. Он предвосхитил появление объектов в привычных языках программирования.

Разработка порождающих слов происходит по-разному в разных диалектах Форта. Язык Каллисто 1.0 (Каллисто Классик) использует симметричный синтаксис `<BUILDS ... DOES>` , известный по ранним версиям языка Форт:

```
: название <BUILDS действия1 DOES> действия2 ;
```

Такая фраза создаёт новое порождающее слово с именем *название*. При исполнении этого слова создаётся новое слово, имя которого указано после порождающего слова.

В Каллисто уже есть встроенные порождающие слова, в частности, стандартное универсальное слово `CREATE`. Как вам уже известно, `CREATE` создаёт новое слово, указывающее на текущий свободный байт в словаре (на момент сразу после создания слова). При исполнении порождённого слова адрес этого байта кладётся на стек.

Конструкция `<BUILDS ... DOES>` расширяет возможности Каллисто по сравнению со словом `CREATE`. При исполнении вашего порождающего слова встроенное слово `<BUILDS` создаёт новое слово, указывающее на текущий свободный байт в словаре. Затем выполняются *действия1,* они же *создающая часть*. А при исполнении порождённого слова адрес этого байта кладётся на стек, а затем выполняются *действия2* (*обработчик*).

Разберём самый простой случай, когда все эти действия пусты:

```
: MYCREATE ( -- )  <BUILDS DOES> ( -- a ) ;
```

Слово `MYCREATE` делает то же самое, что слово `CREATE`, правда, более медленно. В подобном слове нет практического смысла.

В этом примере можно также заметить две стековые диаграммы — тогда как во всех предыдущих примерах было по одной. Для порождающих слов, однако, традиционно пишут две диаграммы. Первая стековая диаграмма, на обычном месте, относится к самому порождающему слову. А вторая, после `DOES>` — к порождаемому слову. В данном случае порождаемое слово кладёт адрес на стек — поэтому его диаграмма выглядит как `( -- a )` .

Однако вспомним: слово `CREATE` создаёт слово, указывающее на свободный байт, но не выделяет какую-либо память. Если требуется массив данных, приходится дополнительно вызывать слово `ALLOT`. А вот при помощи порождающих слов это можно сделать автоматически, чтобы не писать `ALLOT` каждый раз:

```
: CREATEALLOT ( n -- )  <BUILDS ALLOT DOES> ( -- a ) ;
```

Слово `CREATEALLOT` не просто создаёт новое слово, но сразу выделяет память. Количество выделяемых байт памяти нужно положить на стек перед исполнением `CREATEALLOT`. Например, фраза:

```
10 CREATEALLOT ARRAY10
```

создаёт слово, указывающее на выделенный массив из 10 байтов. Она эквивалентна фразе `CREATE ARRAY10 10 ALLOT` , но она куда удобнее, если в программе требуется много массивов.

```
Каллисто Классик 1.0
3834 байт свободно
: MYCREATE ( -- ) <BUILDS
 DOES> ( -- a ) ; ok
: CREATEALLOT ( n -- )
<BUILDS ALLOT DOES> ( --
 a ) ; ok
10 CREATEALLOT ARRAY10
 ok
65 ARRAY10 3 + C! ok
ARRAY10 3 + C@ . 65 ok
```

Чтобы получить доступ к одному из элементов массива, приходится повозиться — например `ARRAY10 3 + C@` кладёт на стек ARRAY10[3], а `ARRAY10 3 + C!` записывает. Но при помощи порождающих слов можно частично автоматизировать и эту задачу:

```
: ARRAY (n -- )  <BUILDS ALLOT DOES> ( i -- a )  + ;
```

Слово `ARRAY` создаёт порождённые слова так же, как `CREATEALLOT` – сразу выделяет байты памяти в количестве, указанном на стеке. Но теперь, если исполнить порождённое слово, оно автоматически берёт со стека значение и прибавляет его к адресу, на который указывает слово. Например, создадим массив из 15 байт:

```
15 ARRAY ARRAY15
```

Теперь, чтобы получить на стеке значение ARRAY15[3], можно применить конструкцию `3 ARRAY15 C@` , а чтобы записать значение со стека в эту же ячейку – `3 ARRAY15 C!` .

```
Каллисто Классик 1.0
3834 байт свободно
: ARRAY ( n -- ) <BUILDS
ALLOT DOES> ( i -- a ) + ;
 ok
15 ARRAY ARRAY15 ok
65 3 ARRAY15 C! ok
3 ARRAY15 C@ . 65 ok
```

Возможности механизма порождающих слов огромны; так, по мнению Баранова и Ноздрунова, он «составляет одно из основных достоинств языка Форт, главную «находку» его создателей». Впрочем, и возможности Каллисто в целом очень заметно шире, чем то, что мы могли описать в кратком введении. Их легче всего изучать в процессе практической работы на Каллисто, сверяясь с литературой.

6 Разработка и сохранение приложений

Когда вы определяете любое слово в Каллисто, вы уже создаёте программу. Вы можете затем вызвать это слово, и программа выполнится.

Однако эта программа существует лишь в оперативной памяти. Стоит выключить ЭКВМ или выйти из Каллисто, и она окажется забыта.

Чтобы сохранить сеанс работы перед выключением МК-161, используйте слово `ВЫКЛ`. Сохранить готовое приложение вы можете, воспользовавшись двумя разными способами.

6.1 Сохранение состояния Каллисто

Можно сохранить на диск ЭКВМ состояние Каллисто полностью, включая все слова, которые определены в Каллисто в настоящий момент. (Строго говоря, сохраняется содержимое словаря и все переменные). Восстановив состояние Каллисто позже, вы сможете пользоваться созданным приложением снова. Чтобы сохранить приложение таким образом, используется слово `SAVE"` . Оно сохраняет состояние Каллисто в файл, имя которого идёт после слова (и после пробела!) и вплоть до ещё одной литеры `"`. В свою очередь, слово `LOAD"` позволяет загрузить это состояние:

```
SAVE" моё_приложение"
LOAD" моё_приложение"
```

Этот способ обладает высоким быстродействием и очень прост в использовании, но есть у него и недостатки. Такое сохранение жёстко привязано к использованной версии Каллисто. Его невозможно загрузить в другую версию (более раннюю, обновлённую или изменённую). Кроме того, исходный текст сохранённого приложения нигде не виден.

Перенести сохранённое состояние на другую ЭКВМ можно, но только если используется та же версия Каллисто.

6.2 Хранение исходного текста приложения на внутреннем диске ЭКВМ

Существует и другой способ сохранения приложений. При помощи редактора, исполняемого в среде Каллисто, вы можете ввести исходный текст своего приложения в *блок* на диске. Блок (экран) Каллисто содержит 48 строк по 64 символа. Экран, находящийся в блоке номер *n*, просматриваются командой `n LIST`, а редактируется *редактором экранов* (см.главу 8). Текст, хранящийся в блоке номер *n*, начинает выполняться и компилироваться при выполнении команды `n LOAD`. На экранах, как и в любых других программах на Каллисто, можно использовать комментарии, закрывая их в круглые скобки:

```
( комментарий)
```

Компиляция на «Электронике МК-161» — процесс долгий. В своём приложении вы можете предусмотреть вывод промежуточных сообщений во время компиляции. Для этого служит слово `.(` — оно похоже на уже известное вам слово `."` (см. 5.1), но употребляется вне определений других слов и выполняется сразу же, во время загрузки и компиляции блока. Чтобы вывести во время компиляции `сообщение`, добавьте в исходный текст фразу:

```
.( сообщение)
```

Важно: не следует путать ввод исходного текста в блоки на диске и определение новых слов. И то, и другое можно назвать разработкой приложений. Но в приложении, находящемся в блоке на диске, можно определить любое количество новых слов и использовать существующие. Также вы можете определить новое слово с клавиатуры («режим пульта»), не записывая его исходный текст в блок на диске.

7 Блоки, хранение исходного кода и его компиляция

В Каллисто блоки занимают по 3 Кбайта и обычно хранят экраны с исходным текстом, 48 строк по 64 символа. Нумерация строк осуществляется с 0 до 47. В строке номер 0 принято хранить *индекс* (заголовок) — комментарий с кратким содержанием экрана. Примерно первые два десятка символов нулевой строки — самые важные и именно они выводятся из соседних блоков с помощью слова `нач кон INDEX`.

На диск МК-161 влезает чуть больше 150 блоков. Стандартное слово `BLOCK ( n -- a)` загружает блок номер n в память и возвращает адрес его первого байта.

Блоки размещаются на диске А в подкаталогах `Каллисто1` `Каллисто2` `Каллисто3` и `Каллисто4` головного каталога. Каждый подкаталог содержит до 60 файлов типа Т (текст) с именами `B001`..`B999`. Если блок или подкаталог не существуют, Каллисто их создаёт:

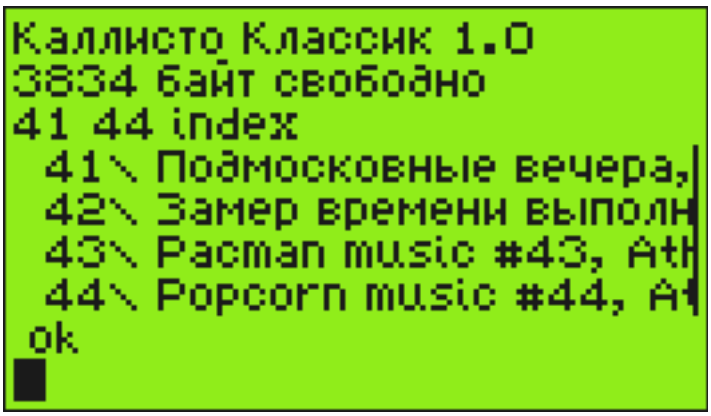
- Каталог `Каллисто1` содержит блоки от B001 до B060
- Каталог `Каллисто2` содержит блоки от B061 до B120
- Каталог `Каллисто3` содержит блоки от B121 до B180
- Каталог `Каллисто4` содержит все оставшиеся блоки

Вывести на индикатор блок можно с помощью слова *n* `LIST`. Загрузить блок и выполнить оттуда слова, в том числе компиляцию новых слов, можно с помощью слова *n* `LOAD`.

7.1 Перечень блоков

- Блоки №1-10 зарезервированы под будущие версии Каллисто. Для совместимости с ними их использование не рекомендуется.
- Блок №11 содержит строковый редактор `EDIT`
- Блок №41 содержит пример «Подмосковные вечера», позволяющий создавать на Каллисто простейшие одноголосые мелодии
- Блок №42 содержит инструмент «Секундомер» для измерения скорости работы фрагментов приложений
- Блок №43 содержит пример «Музыка Pacman»
- Блок №44 содержит пример «Музыка Popcorn» из игры Digger

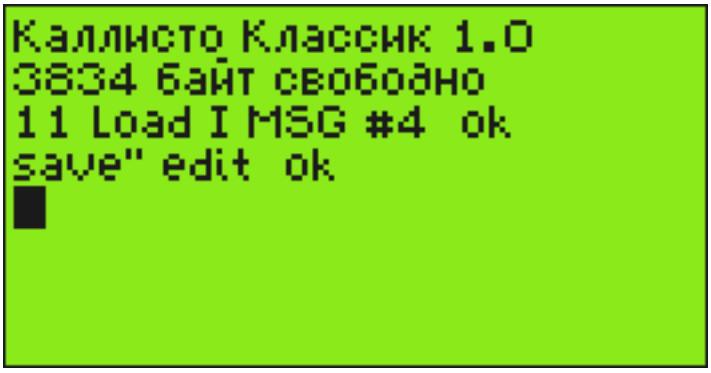
В каталоге `Blocks-compiled` дистрибутива хранятся эти пять приложений, уже откомпилированные.



8 Строковый редактор EDIT

Строковый редактор EDIT предназначен для редактирования 3Кб экранов Каллисто, состоящих из 48 строчек по 64 символов каждая. Строки нумеруются с нулевой (где обычно располагается индекс, справочный комментарий ко всему экрану) по 47-ую.

Стандартное расположение исходного кода редактора EDIT — блок номер 11. Туда он попадёт, когда вы загрузите в МК-161 файл B011.mkt, а потом сохраните его в файл с именем B011 типа T (текст), в каталог Каллисто1 встроенного электронного диска (диск A). Компиляция редактора по команде `11 LOAD` может занять более 15 минут. Чтобы в дальнейшем загружать редактор быстрее, скиньте его на диск сразу после компиляции с помощью команды `SAVE" EDIT`.



Чтобы обойтись вовсе без компиляции, можно перенести в каталог «Каллисто» внутреннего диска уже откомпилированный редактор из поставки Каллисто, состоящий из двух файлов Edit.mkb и Edit.mkd.

Если в описании после команды идёт слово «текст», это означает, что любой текст, набранный после команды, будет скопирован в текстовый буфер, который данная команда использует. Содержимое буфера будет использовано во время выполнения команды. Если пользователь не ввёл никакого текста, во время исполнения команды будет использовано прошлое содержимое буфера, оставшееся от предыдущих команд.

Номер редактируемого экрана берётся из переменной `SCR` — туда его заносят команды `LIST` и `EDIT`, но можно занести и вручную. Все слова редактора располагаются в лексиконе `EDITOR`. Команды редактирования изменяют экран в памяти МК-161, но не на диске. Чтобы сохранить результат редактирования, используйте команду `SV`.

`EDIT` EDIT (*#блока* --)

Приступить к редактированию экрана, расположенного в блоке *#блока*. #блока записывается в переменную `SCR` и текущим контекстом объявляется `EDITOR`.

`SV` SaVe (--)

Записывает отредактированный экран на диск.

`X` eXtract (--)

Копирует текущую строку в буфер INSERT и удаляет её с экрана. Все последующие строчки сдвигаются вверх, строка 47 остаётся пустой.

`T` Type (*n* --)

Вывести строку *n* текущего экрана. Поставить курсор на начало этой строки. Часто используется для задании номера строки перед командами `P X U` и др.

L List (--)

Выводит редактируемый экран на индикатор. Как команда Каллисто `LIST`, только номер экрана берётся из `SCR` — её не приходится набирать. Не меняет позиции курсора.

LS LineS (*нач кон* --)

Вывести строчки текущего экрана с *нач* по *кон*. Команда `LS` не меняет позиции курсора и не выводит его позиции.

N Next (--)

Увеличивает номер текущего экрана на 1. Если старый экран не сохранён по команде `SV`, его содержимое пропадает.

B Back (--)

Уменьшает номер текущего экрана на 1. Если старый экран не сохранён по команде `SV`, его содержимое пропадает.

P Put (--)

P *текст*

Любой идущий следом *текст* будет скопирован в буфер INSERT. Буфер INSERT будет скопирован в текущую строку, заменяя её предыдущее содержимое. Если текст состоит из одного или более пробелов, текущая строка будет стёрта (заменена пробелами).

WIPE Wipe (--)

Стирает текущий экран, забивая его пробелами. Эквивалент команды `CLEAR` (у нас такой пока нет — AtH), только пользователю не нужно вводить номер экрана.

COPY Copy (*откуда куда* --)

Копирует один экран на другой. У нас это команда ядра Каллисто и доступна всегда, даже без загрузки редактора.

I Insert (--)

I *текст*

Любой идущий следом *текст* будет скопирован в буфер INSERT. Команда `I` копирует содержимое буфера INSERT в текущую строчку, начиная с текущей позиции курсора. Любой текст справа от курсора будет сдвинут вправо, за пределы строки и потерян, если общая длина строки превысит 64 символа.

U Under (--)

U *текст*

Любой последующий *текст* будет скопирован в буфер INSERT. Раздвигает экран на одну линию, непосредственно ПОД текущей, вставляя пустую строку. Все последующие линии сдвигаются вниз. Любое содержимое строки 47 будет потеряно. Содержимое буфера INSERT будет скопировано на пустую строку, и эта линия станет текущей строкой.

M Move (*#блока #строки* --)

Копирует текущую строку в буфер INSERT, затем копирует содержимое буфера INSERT в блок, указанный *#блока*, ПОД строкой, указанной в *#строки*. Исходный номер блока восстанавливается, а следующая линия в блоке становится текущей. Это позволяет копировать последовательные линии минимальным количеством нажатий на клавиши. К сожалению, побочный эффект этой команды — чтобы скопировать что-то в строку 0 на другом экране, вам надо вначале скопировать это ПОД строку 0, используя команду `xxx 0 M`, сделать экран *xxx* текущим и затем извлечь (`X`) старую строку 0, пододвинув всё вверх.

^ (--)

Используется как разделитель для любых команд, позволяющих ввод текста — таких как `P` и `U`. Позволяет вводить более одной команды на той же самой строчке, например:

`3 T P` Это строчка `3^ L` (ввод)

Хотя эта особенность полезна, она не позволяет использовать «^» как символ в любом тексте, который записывается на экран.

9 Главные отличия Каллисто от стандартного Форта

Вместо циклов DO LOOP используется взятый из colorForth цикл FOR NEXT (см. 5.6)

Для команд DUP SWAP INVERT и NEGATE используются традиционные для ПМК имена ↑ ↔ NOT и /-/.

Стек данных (см. 5.2, 5.3) содержит десятичные числа с плавающей запятой «Электроники МК-161». Если целые числа по модулю меньше 1000 миллиардов (10^{12}), потерь значащих цифр в стеке из-за этого не происходит.

Для ввода дробных чисел используется десятичная запятая. Набор слов для целых чисел двойной точности сильно ограничен, ведь в одной ячейке стека с лёгкостью помещаются 32-битные двоичные числа и 12-разрядные десятичные числа.

При вводе знака числа или порядка нельзя использовать +, только − либо ничего.

При компиляции можно использовать целые числа только в диапазоне от −32768 до 32767. Числа за пределами этого диапазона должны содержать десятичную запятую.

Тип чисел в стеке и памяти десятичных данных (единые адреса 10000…10999) один и тот же, но различен при размещении данных в двоичной памяти (единые адреса 0…9999, а также 11000 и больше). Для считывания беззнаковых 16-битных целых служит уникальное слово U@ (см. 5.8).

Для умножения и деления с плавающей запятой используются символы ПМК × и ÷. Символ / используется для целочисленного «фортовского» деления с остатком, символ * в стандартных словах не используется.

Слово NUMBER оперирует с простой ячейкой стека, а не двойной.

Слово . (точка) умеет работать с плавающей запятой, если активна десятичная система счисления. Слова Форта-2012 . и .R переименованы в I. и I.R.

Слова поиска 'N FINDN возвращают NFA, а не CFA. Используйте слово NAME> для получения CFA. Слово FINDN пока сохраняет отпечаток Форта ИТЭФ.

Вместо CREATE DOES> из Форта-2012 используется <BUILDS DOES> из FIG-FORTH и дополнения к FORTH-79 (см. 5.9). Различия минимальны, просто используйте <BUILDS вместо CREATE, когда компилируете порождающее слово.

Длина строки, возвращаемая (LINE), указана с учётом убранных с помощью -TRAILING пробелов в конце строки.

9.1 Стандартные слова Каллисто, которые могут быть плохо знакомы новичкам

ACCEPT из Форта-2012 используется вместо устаревшего EXPECT (см. 11.1).

VARIABLE не требует числа на стеке для инициализации переменной, начиная с Форта-79 (см. 5.4).

Слово FOR берёт n с вершины стека и исполняет цикл FOR NEXT ровно n раз. Если n⩽0, управление сразу передаётся на код за словом NEXT. Внутри цикла слово I показывает, сколько ещё раз циклу выполняться, включая текущий раз. Слово LEAVE покидает цикл сразу же (см. 5.6).

Рекомендуется использовать порождающее слово VALUE вместо VARIABLE (см. 5.4). Когда вам нужно хранить переменные с плавающей точкой или целые числа, большие по модулю, оно необходимо.

Для ввода чисел в других системах счисления (отличных от BASE) можно использовать префиксы Форта-2012: $1a #99 %1101 'c' (см. 5.2).

9.2 Слова, существующие только на Каллисто

Для прямого доступа к регистрам «Электроники МК-161» служат слова П ИП. Трансляции адресов не происходит, например 13 ИП считывает на стек R13, который ещё называют RД.

Десятичные регистры организованы в *десятичный словарь (Д)*, которым управляют слова ДН ДHERE ДALLOT Д, (см. 5.8).

Целый ряд слов, таких как sin cos lg ln 10^x e^x √ x^2 π СЧ ЗН [x] {x} и т.п. взят из ЯМК (см. 5.2).

Слово ГРФ обновляет графический экран, отображая нарисованную картинку.

Слово BEEP издаёт гудок заданной частоты и продолжительности, а слово (PLAY) проигрывает мелодию из заранее подготовленного массива.

Слова U# U#S U#> оперируют с простой ячейкой стека, а не двойной. Технически, слова <# SIGN уже соответствует словам Форта <# SIGN. Указанные три легко переопределить до стандарта, но лучше наоборот — переписывать определения, обращающиеся к этим удобным словам.

Слова <F# F# F#> осуществляют форматное преобразование не только целых чисел, но также десятичных дробей и чисел с плавающей запятой.

9.3 Таблица слов Каллисто, не имеющих аналогов в Форте

(PLAY)	(a u --)	Проиграть мелодию из u нот по адресу a (см. 9.2).
APP	(-- a)	Переменная автозапуска после LOAD" (см. 6.1).
INKEY	(-- c)	Опрос клавиатуры (см. 11.1).
LOAD"	(--)	Загрузить словари с диска (см. 6.1).
SAVE"	(--)	Сохранить словари на диск (см. 6.1).
TYPE1	(a u --)	Вывести в одну строку индикатора не более u символов, начиная с адреса a
ГРФ	(--)	Обновить графический экран (см. 9.2, 11.2).
Д,	(x --)	Записать число в очередной десятичный регистр (см. 9.2).
ДALLOT	(u --)	Зарезервировать u десятичных регистров (см. 5.8, 9.2).
ДН	(-- a)	Переменная, хранящая вершину десятичного словаря (см. 9.2).
ДHERE	(-- u)	Номер первого свободного десятичного регистра (см. 5.8, 9.2).
ИП	(u -- x)	Прочитать регистр номер u (см. 9.2).
П	(x u --)	Записать x в регистр номер u (см. 9.2).
(ИП)	(-- r)	Считать значение регистра, зашитого в шитом коде

10 Сообщения об ошибках

При ошибке Каллисто выводит на индикатор номер ошибки и переходит в пультовый режим. Предупреждения также имеют номер ошибки, выводимый на индикатор, но остановки выполнения программы не происходит.

MSG# 0	Слово не узнано.
	Число не узнано.
	Нет соответствия системе счисления
MSG# 1	Попытка извлечь нечто из пустого стека
MSG# 2	Переполнение стека или словаря
MSG# 4	Повторное описание слова (не является фатальной ошибкой)
MSG# 17	Используется только при компиляции
MSG# 18	Используется только при исполнении
MSG# 19	`IF` и `THEN` или другие операторы не имеют пары
MSG# 20	Определение не завершено
MSG# 21	Неверный аргумент слова `FORGET` (нет в Каллисто 1.0)
	Слово в защищённой области словаря
MSG# 22	Должно использоваться только при загрузке
MSG# 23	Строка экрана вне диапазона 0..47
MSG# 24	(нет в Каллисто 1.0)
MSG# 26	Деление на 0 (нет в Каллисто)
MSG# 39	Сохранение от другой версии
MSG# 40	Файл не найден
MSG# 41	Нет диска
MSG# 42	Диск не форматирован
MSG# 43	Нет места на диске
MSG# 44	Нет места в каталоге
MSG# 45	Ошибка имени файла/каталога
MSG# 46	Невозможно удалить/создать/загрузить файл/каталог
MSG# 47	Файл/каталог не выбран
MSG# 48	Нет разрешения дисковой операции

11 Внутренние и дополнительные возможности Каллисто

11.1 Ввод и вывод

Мы уже знакомы с простыми способами вывода информации. Слово `.` выводит на экран число с вершины стека, а слово `."` используется для вывода текста (см. 5.1 и 5.2).

Ввод информации, как таковой, в программах на Форте обычно не реализуют. Вместо этого, чтобы обработать те или иные числа, пользователь кладёт их на стек, а затем вызывает определённое в программе слово.

Если же нужно ввести строку, она, напротив, вводится после определённого программой слова.

Каллисто предоставляет дополнительные возможности ввода-вывода на случай, если требуется работать напрямую с клавиатурой и экраном ЭКВМ.

Слово KEY отображает курсор, ожидает ввода литеры с клавиатуры и кладёт на стек код введённой литеры, число от 0 до 255. Слово EMIT берёт код литеры со стека и выводит литеру на индикатор. Слово TYPE берёт из стека адрес и длину строки, после чего выводит её на индикатор. Это всё — стандартные и давно устоявшиеся слова Форта.

С «сырой» клавиатурой МК-161 работают два слова EKEY INKEY. EKEY ожидает нажатия на клавишу МК-161 и кладёт на стек код этой клавиши, число от 0 до 37. INKEY кладёт на стек код нажатой клавиши или −1, если никакая клавиша в данный момент не нажата. KEY EKEY последовательно используют буфер клавиатуры МК-161, а INKEY может не заметить нажатие клавиши, если пользователь уже отпустил её к моменту проверки.

Для ввода с клавиатуры строки символов служит слово ACCEPT. Параметры у него такие же, как у слова TYPE, а возвращает оно на стеке число введённых символов, которое может оказаться меньше запрошенного. При вводе можно использовать клавиши редактирования `BS (Cx)` и `F BS` (см. главу 4).

11.2 Машинная графика

Программа на Каллисто может «рисовать» на экране микрокалькулятора. Доступны два цвета — чёрный и белый (точнее, цвет подсветки — обычно зелёный).

Координаты исчисляются от точки (0,0) в верхнем левом углу экрана. Размер экрана — 128 x 64 точки.

Вы можете использовать приведённые ниже слова для «рисования». Однако «нарисованное» не отображается на экране сразу. Для отображения следует исполнить слово `ГРФ`.

`AT`	(x y --)	задать начальную координату (позицию графического курсора)
`+BOX`	(dx dy --)	закрашенный прямоугольник
`+FRAME`	(dx dy --)	рамка
`BAR`	(x y --)	отрезок от AT до (x,y)
`DOT!`	(x y --)	установка точки
`DARK`	(--)	устанавливает чёрный цвет (рисует)
`LIGHT`	(--)	устанавливает белый цвет (стирает)
`ATR!`	(c --)	установить атрибут вывода на индикатор c (0..7)

11.3 Как Каллисто использует регистры МК-161

R0 временный регистр слова-примитива
R1 временный регистр слова-примитива
R2 RP указатель целочисленного стека возвратов (RS) —
растёт вниз от 5090
R3 SP указатель десятичного стека (DS с плавающей запятой) —
растёт вниз от 998

R4 временный регистр слова-примитива
R5 временный регистр слова-примитива,
может быть изменён NEXT, CALL, EXIT
R6 RI указатель шитого кода, он же IP указатель инструкций,
указывает на следующий CFA (когда R9==NEXTD)

R7 W указатель слов Форта (WP), содержит CFA исполняемого слова,
слово-примитив может менять R7
R8 JP указатель/регистр передачи управления NEXT,
временный регистр слова-примитива
R9 =NEXTP/NEXTD, адрес кода NEXT
RA временный регистр, его может изменить BIOS
RB временный регистр слова-примитива
RC =RPUSHRIP/RPUSHRID
адрес кода сохранения указателя шитого кода RI в стеке возвратов
RD при загрузке/сбросе словаря содержит номер версии Каллисто
RE =256, литерал
R15 RX Здесь сохраняется стек ЭКВМ при считывании регистра словом ИП
R16 RY
R17 RZ
R18 RT
R19 9007/9008 (DARK/LIGHT)
номер регистра прокрутки для текущего цвета фона

R9042 RI указатель шитого кода (когда R9==NEXTP)
R9044 читает байт шитого кода, RI++

11.4 Единое адресное пространство

Единая адресация Каллисто объединяет пространства регистров и памяти программ так:

0..9999	байты памяти программ, только для чтения -- система ввода-вывода, словарь Форта (начало)
10000..10999	десятичные регистры (R0…R999):
10000..10019	регистры Форта, служебные и зарезервированные
10020..ДHERE-1	десятичный словарь
SP..10997	стек данных (параметров),
11000..15095	двоичные регистры (4 Кб, R1000…R5095):
11000..11095	входной буфер терминала (TIB)
	и признак конца ввода (2 байта),
11096..11142	область переменных Форта (USER),
11143..HERE-1	словарь Форта (продолжение),
HERE..HERE+84	буфер форматированного вывода (PAD),
RP..15089	целочисленный стек возвратов,
15094..15095	номер блока, загруженного в буфер
	и его флаг сохранения (2 байта)
15096..18167	область текста,
	буфер с исходным текстом для компиляции (3 Кб, R5096…R8167)
18168..18999	не используется (всегда 0)
19000..19999	регистры функций «Электроники МК-161»

Дно стека записано в переменной `S0`, а дно стека возвратов — в переменной `R0`.

12 Хранение чисел в памяти

Стандарт размещения многобайтных слов: в двоичном виде, старшее по младшему адресу.

Операции с беззнаковыми 16-битными числами добавляют к отрицательным аргументам 65536.

Операции с беззнаковыми 32-битными числами добавляют к отрицательным аргументам 65536^2 = 4 294 967 296.

13 Словарная статья

Каллисто Классик использует косвенный шитый код. Определения через двоеточие состоят из последовательности скомпилированных CFA — указателей на поля кода используемых слов. Токен исполнения xt представляет из себя CFA — адрес поля кода соответствующей статьи.

Адрес в самом поле кода может указывать только на ЯМК в памяти программ.

Словарная статья состоит из 4 полей: *Name Link Code Parameter*

Адреса этих четырёх полей (в единой адресации) называются NFA LFA CFA PFA

Заголовок статьи это первые два поля: *Name Link*

После заголовка идёт тело статьи: *Code Parameter*

где *Name* это поле имени: *флаг-счётчик имя*

Link это поле связи (2 байта) — NFA предыдущего слова

Code это поле кода (2 байта) — ссылка на интерпретатор слова, то есть исполняемый код ЯМК в памяти программ

Parameter (размер не фиксирован) это поле параметров, реализация слова: шитый код, код на ЯМК, область данных и т.д.

Поисковые слова Каллисто дают NFA, откуда путём преобразований получаем адреса любых других полей. Обратная операция, получение NFA слова по адресу других полей, в Каллисто невозможна.

Флаги в первом байте поля имени: $80 IMMEDIATE, $20 SMUDGE (HIDDEN), $40 не используется.

14 Решение проблем

После первого запуска Каллисто с моей «Электроникой» происходит что-то непонятное. Клавиатура заблокирована, повторная запись любых mkp невозможна. Не помогает ни перезагрузка, ни перепрошивка внутренней программы. Моя ЭКВМ превратилась в кирпич, я в панике, помогите!

Убедитесь, что заряда аккумулятора вашей «Электроники МК-161» хватает для автономной работы. Выключите ЭКВМ и отсоедините её от зарядного устройства.

Через 10 секунд нажмите клавишу С/П (литеры ; С на накладной клавиатуре) и включите вашу «Электронику», удерживая эту клавишу нажатой. Это должно отменить автозапуск Каллисто и решить указанную проблему. (Подробнее см. главу 4).

Перед повторным запуском Каллисто внимательно изучите настоящее Руководство. Работа «Электроники МК-161» под управлением Каллисто радикально отличается от обычного режима этой машины и требует навыков, описание которых отсутствует в документации производителя.

Что делает такое-то слово Каллисто?

Краткое, в одну строчку, описание каждого встроенного слова Каллисто можно найти в файле words.txt из каталога docs.

Если краткого описания недостаточно, зайдите не сайт http://pmk.the-hacker.ru/ , где поддерживается Вики по ЭКВМ и ПМК серии «Электроника». На Вики размещена подробная информация о каждом встроенном слове Каллисто, включая цитаты из стандартов разных лет, примеры реализаций и ссылку на исходный текст каждого из слов, а также информация об основных особенностях архитектуры Каллисто и «Электроники МК-161».

Указанная Вики не имеет отношения к производителю «Электроники МК-161» и содержит информацию непосредственно от разработчика Каллисто Классик. Вики по ЭКВМ доступна для редактирования. Если вы замечаете ошибки на её страницах, не стесняйтесь их исправлять. Регистрация на Вики поможет другим участникам общаться с вами и относиться серьёзнее к вашим исправлениям и добавлениям.

15 Синтаксис Каллисто Классик в РБНФ

Программа = { ОбъявлПерем | ОбъявлСлова | Команда }.

ОбъявлПерем = (**VARIABLE** | **FVARIABLE** | **CONSTANT** | **VOCABULARY** | **VALUE** | **DEFER**) Слово.

ОбъявлСлова = ':' Слово ПослОператоров [ОпределяющееСлово] ';' [**IMMEDIATE**].

ОпределяющееСлово = '<BUILDS' ПослОператоров 'DOES>' ПослОператоров.

ПослОператоров = { Оператор }.

Оператор = УсловныйОператор | ЦиклОператор | Команда.

Команда = Комментарий Текст ")" | КомандаСтр Текст '"' | ("'" | "'N") Слово | **TO** Слово | '[COMPILE]' Слово | Слово | "'" Литера "'" | Число.

Комментарий = '(' | '.('.

КомандаСтр = '."' | 'LOAD"' | 'SAVE"'.

УсловныйОператор = **IF** ПослОператоров [**ELSE** ПослОператоров] **THEN**.

ЦиклОператор = **BEGIN** ПослОператоров КонецЦикла | **FOR** ПослОператоров **NEXT**.

КонецЦикла = **AGAIN** | **UNTIL** | **WHILE** ПослОператоров **REPEAT**.

Число = [Префикс] ["-"] { цифра } ["," { цифра }] ["E" Порядок].

Префикс = "#" | "$" | "%".

Порядок = ["-"] цифра [цифра].

Цифра = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "A" | "B" | "C" | "D" | "E" | "F".

Текст = Литера { Литера | " " }.

Слово = Литера { Литера }.

```
: START
  ." HELLO, WORLD!"
;
```

МОСКВА·2017