

Pure C — Lurkmore

По [техническим причинам](#) запрос «C#» перенаправляется сюда.

C (Си) — язык программирования, разработанный придуманный по приколу в [расовой пиндосской](#) компании Bell Labs в начале 1970-х годов Деннисом Ритчи. Является на сегодняшний день фактически самым низкоуровневым из языков высокого уровня, и, как следствие, предоставляет достаточно гибкие возможности по использованию ресурсов компьютера благодаря повсеместному использованию указателей и операторов приведения типа. Но на том всё и заканчивается: ООП, динамика, метапрограммирование — всё это реализовали в родственниках и потомках. Среди [программистов](#) носит неофициальный титул «кроссплатформенного [ассемблера](#)». Ответственность за корректную работу программы целиком и полностью лежит на программисте, за что Си ненавидим [быдлокодерами](#) и, что важно, их начальством. Хорошо мотивированного project manager'a, писавшего когда-то в патлатой молодости на [Java](#), легко можно ввести в ступор, предъявив часть проекта на Си.

Быдлокод на Си обычно [чуть более, чем полностью](#) состоит из [переполняющихся буферов](#) и битья памяти, а также является излюбленной мишенью для экспериментов [кулхацкеров](#).

Откуда ноги растут

Давным-давно в далекой, далекой галактике, существовала ЭВМ под названием PDP-11. Именно под эту машину перед Деннисом Ритчи и стояла задача создать новый язык программирования, пригодный для написания в том числе и ядра [операционной системы](#). И как раз системе команд PDP-11 язык C обязан массой своих синтаксических особенностей. Все эти [пре- и пост-инкременты/декременты](#), нативная поддержка null-terminated строк, и прочие, хорошо знакомые любому C-программисту концепции, практически напрямую взяты из системы команд PDP-11. В итоге для PDP-11 сабжевый язык оказался по сути макроассемблером, где каждый оператор ложился на логически завершённый [набор инструкций](#). Что касается [остальных аппаратных архитектур](#), то здесь, вопреки заявлениям дилетантов, сабж никакой особой низкоуровневыми не отличается, ведь адресная арифметика и доступ к памяти и портам ввода/вывода (в системах, где это вообще разрешено) есть в любом мало-мальски юзабельном [ЯП](#), от [Фортана](#) до [Паскаля](#). В итоге по факту на сегодняшний день C — ещё один быдлокодерский язык высокого уровня, ничем принципиально не отличающийся даже от некоторых продвинутых реализаций [BASIC](#)-а, так-то!

Мёртвый язык

[Многие](#) почему-то считают, что Си мёртв. Когда-то на нём писался практически весь софт, и понятие «быть программистом» однозначно и безальтернативно включало в себя «знать Си». Сейчас, конечно же, это не так, и Си успешно вытеснили практически отовсюду. Но

Научно-технический рэп -
[Папа может в си](#)
Папа может в C

красноглазики не унывают, и до сих пор многие [кошерные](#) софтинки (зачастую с весьма навороченным пользовательским интерфейсом) пишутся на голом Си (Wireshark, например). Но из-за того, что практически единственным вменяемым для такого рода джедайства фреймворком является GTK (или Eclipse), на подвиг отправляются исключительно закоренелые [гномосеки](#). Си никогда не умрёт, пока есть микроконтроллеры и нужно писать драйвера.

Есть, конечно, некоторые проблемы. Бородатые олدفаги, единственные, кто умеет писать ядра операционки, драйвера и системные службы, демонстративно отказываются учить что-то ещё. [На самом деле](#), любой уважающий себя сишник знает не только C++, но и ещё с десяток других языков, главным образом для того, чтобы их обсирать. А на Си они любят писать потому, что код в таком случае получается короче и прямее. Поэтому ядра и драйвера в их пространстве пишутся только на Си, и ни одного на [Perl](#)'е и, тем более, [PHP](#) — ну не пихать же интерпретатор прямо в ядро. Это злостный цинизм и несправедливая конкуренция. А системные службы написаны на Си [чуть менее, чем все](#). Кое-кто ехидно предполагает, что наличие хоть какого-нибудь вменяемого exceptioning'a в Си могло бы исключить появление [BSOD](#) в 90-х. Но [всем похуй](#).

Си и объекты

В середине 80-х некто Страус Трупп (наст. имя Bjarne Stroustrup, не путать с Леви Страуссом) решил продвинуть дальше идею кросскомпилируемого ассемблера. [Бородачи](#), не сподвигнувшиеся выучить ООП ^[1] (а некоторые из них про него и не слышали, искренне считая все технологии после 1981 г. унылым говном), разжигают холивар, который, как правило, сводится к необходимости описывать каждый чих в ООП с одной стороны и необходимость изобретать уйму велосипедов для хоть какой-нибудь объектности с другой. Однако, как показывает практика, там, где задача предрасполагает к объектности сама по себе, не использовать готовый инструментарий в подавляющем большинстве случаев глупо.

Плюсики в названии C++ (Си Плюс Плюс, иногда *CPP*) обозначают наличие в нём объектно-ориентированного программирования, реализованного, правда, с учётом местной специфики.

- Вообще-то исконные правила ООП, в том числе и алгоритмические, предполагают полную инкапсуляцию объектов. Объект должен сам решать, что делать когда его что-то попросят, а не выставлять наружу публичные методы, которые дёргает всякий, кому не лень. Настоящий инкапсулированный объект должен принимать снаружи сообщение, причём не в контексте вызывающего объекта, а в своём собственном. Потом думать, хочет ли он это сделать, делать это и в ответ посылать сигнал о результате действия. Хотя многим нравится, но это уже тема отдельного холивара.

Любопытно, кроме C++ у Си есть ещё один обратно-совместимый родственник: **Objective-C**, в котором попытка реализовать объектно-ориентированное программирование до конца увенчалась успехом. Яблочники даже на нём пишут свои поделия (в том числе и интерфейс к [iPhone](#)). Но почему-то отклика в [массах](#) это не нашло.

Си отличается крайней шустростью (быстрее только ассемблер и за столетия допиленный до совершенства фортран), то есть, конечно, шустростью отличаются программисты. [Гений](#) всегда готов написать на Си или асме так, что будет тормозить на любом самом быстром кластере. И Си предоставляет ему в этом просто невероятное море возможностей. Например, возможность невозбранно [выстрелить себе в ногу](#). Только для этого надо указать на участок памяти, где лежит нога, по смещению наложить структуру, пройти по её полям и прямым преобразованием типов (динамического тут нету) передать данные функции «выстрелить». Если что-то произойдёт не так, дадут циферку с номером ошибки. Или не дадут, если функция void. Да, try-catch конструкций тут тоже нет. Ну, то есть, если вы, конечно, хотите, то есть long jump... и даже вроде как есть библиотеки с готовыми реализациями исключений а-ля C++.

[Тысячи их](#), и все [говно](#).

Лютая, бешеная ненависть

ТруЪ Си [люто](#), [бешено](#) любим многими, но так же люто, бешено ненавидим еще более многими. Похоже, что середины здесь нет и никогда не будет. Некоторые предполагают, что водораздел по этому вопросу проходит между теми, кто умеет писать на Си, и теми, кто хотел бы уметь, но не умеет.

Для начинающих можно упомянуть тот факт, что в С (и во всех производных языках) 1/3 будет равно 0.

- Да, выражение $1/3==0$ истинно, хотя $1/3.0==0$ — нет. Но деление через "float"-дробь таит [свои](#) опасности...

При этом `==` как оператор равенства и знак равенства *как оператор присваивания* до сих пор взрывает неподготовленный специальным образом мозг чуть менее, чем напрочь, по ходу и в наше время являясь источником массы трудноуловимых ошибок у умников, которые не искореняют и даже не читают варнинги, по причине использования буржуйскоязычной версии, например.

Также доставляют строки. Со строками в обычных языках ничего не имеют общего, все функции работы перекочевали прямо из [ассемблера](#).

- По умолчанию С-шная строка — это указатель `char*` на первый символ массива символов. Когда с ней надо что-то сделать, стандартные функции просто берут все ячейки от первой, до той, где будет следующий символ `\0`.
- Это порождает ту самую ошибку переполнения буфера (хакеры мстительно передавали на вход длинные строки, оставляя конец массива без нолика, и наивный робот грузил в строку всё подряд, пока не добирался наконец до конца), а также лютую путаницу с кодировками — шарп очень-очень долго был однобайтным. Кто пытался с этим что-то сделать, тот знает, почему «если в C++ в конце концов не появится стандартного класса строк, то на улицах прольется кровь».

MOAR HATE

Привыкшие к строкам и операторам професси-аналы люто, бешено ненавидят сабж по целому ряду более интересных причин.

Во-первых, потому, что отладка в нем может быть сильнейшей еблей мозгов, особенно когда что-либо вытекает из своих границ "вещи в себе" и, постепенно интегрируясь, естественно входит в границы чего-то другого. Проблема зелёных падаванов, не имеющих ещё даже первых 10 лет опыта и ещё не постигших истин дзен-отладки. Ибо это вам не ява, здесь надо головой уметь. Сейчас (на самом деле давно уже) стало полегче, а в старые добрые DOS-овские времена запуск некошерного кода зачастую приводил к полному зависанию ящика с необходимостью нажимать «Any Key» и ненажатие кнопки «Save» (в общем-то, кнопки тогда были не везде и обычно надо было давить Ctrl+S, F2 или чего еще) до запуска оной некошерной программы каралось ее некошерным перенабиванием и перепрограммированием в кошерную. С нуля. Снова. Да.

Кроме выхода за границы недозволенного, была еще такая штука, как утечка памяти. Это вообще не ловилось никакими отладчиками, и нужно было долго (иногда очень) и вдумчиво (иногда очень) вчитываться в текст такой гадской программы, чтобы понять, куда эта блядская память течет^[2]. Опять-таки теперь сильно полегчало. И не потому, что професси-аналы стали круче, ба потому, что памяти стало в 9000 раз больше и сейчас никого ниибёт, ну и до кучи такая штука родилась, "[Песочница](#)" зовётся. А вот

в старые DOS-овские времена любой профи легко мог доказать, что [640К не хватит на всех](#).

Следующая замечательная вещь — [рекурсия](#). И если продвинутый погромист мог даже объяснить, как считается **C(m, n)** рекурсивной функцией, то отладить рекурсивную прогу из более, чем трех строчек было выше его погромистских возможностей. Хороший, годный компилятор умеет оптимизировать [хвостовую рекурсию](#), но сам язык этого не гарантирует.

Ну и, наконец, **указатели**. Просто терминальный песец/судьбы.

- Если кто-то сомневается или считает, что рекурсия вставляет глубже, можно показать нижеприведённое, [например](#):
 - `double (*(f)(double*)(double)))(double)` — указатель f на функцию, принимающую указатель на функцию, принимающую и возвращающую действительное число, возвращающую указатель на функцию, принимающую и возвращающую действительное число.
 - `int (**f)(char *c)` — двойной указатель на функцию, принимающую строку и возвращающую целое число
 - `int *(f)(char *c)` — указатель на функцию, возвращающую указатель на целое
 - библиотечная `void (* signal(int __sig, void (* __func)(int))) (int)` из `signal.h` возвращает... указатель на функцию.

В стандарте предусматривается **произвольная вложенность** подобного матана, причем объясняется довольно просто. Наличие гибкого механизма использования указателей вообще и указателей на функции в частности, при грамотном подходе позволяет реализовывать крайне элегантные технические решения, совершенно [несопровождаемые экспрессии](#) в дальнейшем.

Такая лютая, бешеная ненависть к фундаментальным понятиям языка не могла пройти незамеченной всякими Майкрософтами. Венцом мелкософтовской ненависти к труЪ Си является появление [C#](#), который на самом деле собственная версия [Жабы](#).

На самом деле, детишки, так устроен ваш ЦП, что поделаться. Что такое функция? Это набор последовательных инструкций, который лежит в виде кучки байтов где-то в памяти (про то, что память виртуальная и лежать он может, например, на диске мы говорить не будем). А раз так, то для процессора все функции выглядят как номер байта, называемый *адресом*, начиная с которого хранится функция (про то, что адресация виртуальная и две программы по одному адресу могут обращаться каждая к своей функции, мы тоже не будем). Этот адрес (который просто число) и есть страшный «указатель на функцию». Некто [Джозель Спольски](#), который на всё горазд, и программист, и программист, утверждает, что этот абзац делит людей на две группы: у одних есть та часть мозга, которая отвечает за понимание указателей, а у других её [нет](#). Анонимус же, напротив, считает, что понимать тут нечего, всё очень просто, но не всем везёт прочесть хорошее объяснение. [Тебе](#) повезло!

Отсюда же растут яйца у произвольной вложенности. Законы природы такую вложенность ничем не ограничивают. Любое же *искусственное* ограничение вложенности будет не менее произвольным. На самом-распределённом деле ограничения есть (во всяком случае, были во времена первых компиляторов), и это следует хотя бы из максимальной длины строки твоего кода. Но на практике тебе указатель на указатель на указатель на указатель просто [не нужен](#).

Страшные записи с херовой прорвой звездочек на практике используют не чаще, чем математики — формулы со 100500 переменными. Как и математики, программисты *группируют* переменные. (В данном случае, в роли переменных выступают *типы*). Проще говоря, тип «указатель на функцию, которая принимает число и возвращает да/нет» обозначают каким-нибудь словом, например `POINTER_TO_PREDICATE`^[3]. В дальнейшем, от типа `POINTER_TO_PREDICATE` можно получить производные типы и дать им тоже понятные и человеко-читаемые имена. Так все, собственно, и делают.

Наконец, возвращаясь к теме ЦП. Си недаром называют «межпроцессорный ассемблер». Это самый низкоуровневый язык из всех высокоуровневых. Хочешь работать с процессором без посредников? Си — твой выбор. Удобнее не влезать в каждую бочку затычкой, а описывать программу «в общих чертах» — бери что-нибудь высокоуровневое, с автоматическим управлением памятью и ресурсами.

Переполнение буфера

За техническими тонкостями просим курить педивикию, здесь же напомним суть и почему эта суть является катастрофой.

Как уже было сказано, в С реализовано прямое управление памятью. А строка, согласно ассемблерной реализации, — это просто указатель типа `char*` на массив байтов, от первого и до тех пор, пока не встретится 0. Все эти функции с названиями, больше похожими на имена древнешумерских демонов (`strcpy`, `strcat`, `atoi`, etc) попросту вызывают одноимённые ассемблерные команды для манипуляций со строками.

Это значит, теоретически, что программа имеет полное право изменить любой участок собственного кода и данных, когда все это висит в памяти. Причем код и данные в памяти располагаются вперемешку. И вот, допустим, программа принимает от анонимуса 10 байтов распознавания [капчи](#). А в 11-м байте ВНЕЗАПНО

должна находиться и "находится" инфа для процессора, какой код выполнять дальше.

Пока анонимус вводит столько, сколько скажут -- 10 или меньше байтов, все хорошо. Но если на месте анонимуса окажется кулхацкер, вводящий вместо капчи 10 байт своего кода, плюс 11-й байт с указанием процессору начать выполнять этот код (уже невозбранно попавший в память, предназначенную для капчи), а С-погромист прозевал проверку на длину получаемых и записанных в память данных — то всё, кулхацкер получил доступ к программе и теперь может легко заменить **весь** ее код своим (в том числе запросив у компьютера еще хоть гигабайт памяти на дополнительные нужды, благодаря все той же прямой работе с памятью).

Особенно кошерно получается, если дырявая программа исполняется из-под [корневого пользователя](#) и ОС позволяет ей вообще все, вплоть до [форматирования самой себя](#), что было совершенно нормальным явлением для всех ОС от Некрософта [до XP](#) включительно. Впрочем, в Висте и Топоре работа с правами настолько уебищна для пользователя, что разграничение прав многие просто отключают, с предсказуемым результатом. А теперь вспомним, что чуть менее чем все ОС и другие популярные и общеупотребимые программы написаны на C/C++...

Короче говоря. Причиной появления 99% дыр в программах, а значит, и всех вирусов и троянов, которые их используют, является то, что эти программы написаны на языке С и прочих низкоуровневых языках с прямым доступом к памяти.

С более современными (или просто написанными без возможности управления памятью) языками, вроде Явы, Сишарпа, [Перла](#), [Питона](#) и даже ПХП устроить подобную дыру невозможно, как правило, даже теоретически — ну разве что она окажется внутри самого компилятора/интерпретатора. Ну да за этим намного легче уследить, чем за over 9000 программ, написанных на С-подобном. С другой стороны, у подобных языков есть другие проблемы и другие виды дыр.

Увы, но Pure C все еще популярен, особенно для написания системного кода, поэтому нас всех ждет еще много веселых и радостным (для хэккеров) дней.

Фауна

gcc

Ярчайшим представителем является его винраршество GCC (вернее, правда, будет «gcc»). Изначально название расшифровывалось как GNU C Compiler, однако по мере скрещивания ужа с ежом, то есть добавления фронт-ендов к другим [языкам программирования](#), оперативно был переименован в GNU Compiler Collection, что не есть [Б](#). Многие поколения красноглазых (и не очень) верующих внесли свою лепту. А первая версия была написана самим [Пророком](#) лично. Так-то.

Отличительной особенностью является то, что именно этот компилятор в 95% случаев является вообще самой первой программой, выполненной свеженаписанным ядром любой свежевывсранной ОС. И только если ядро выполняет без проблем бинарники, собранные gcc, ОС вообще может в принципе считаться годной и, таким образом, переходит в стадию Self-hosting (то есть из-под неё можно хоть как-то в принципе работать, написать первый драйвер, к примеру). Точнее, в случае если gcc не идёт, допиливают именно ОС, а не компилятор, что кагбэ [намекает](#) нам на качество и надёжность кода, им производимого. Таким образом найти ОС, на которой он не пойдёт, намного сложнее, чем жизнь на этом вашем [Марсе](#), нередки случаи, когда вся ОС вместе со всей периферией в полном составе писалась на С при помощи него родного. За это любим всеми сведущими в С.

Исторически gcc всегда представлялся как нечто громоздкое и неуклюжее, генерирующее не самый быстрый код. Но где-то в 2017 году на команду разработчиков нахлынули энтузиазм и просветление, и теперь gcc стал одним из лучших компиляторов. По скорости генерируемого кода он догнал Intel C Compiler, оставив позади и [MSVC](#), и clang, и менее известные компиляторы. Epic win.

clang

Попытка сделать современный gcc без огромного количества legacy-кода, накопившегося для поддержки забытых систем на забытых платформах. Создаётся совместно Google, Apple и много кем ещё.

Больше, чем С

Языку уже больше 40 лет, и целые блоки кода стали в нём 100% стандартны. Были вполне многообещающие попытки оформить синтаксис С в виде ISO-стандарта, так оно каноничнее выйдет. Также было немало попыток сделать языки, которые компилируются в С, который компилируется уже в [Assembler](#). Наиболее эпичны и значимы:

- **C++** — самый-самый первый компилятор Cfront от Страуструпа именно так и работал: брался код на тогдашнем C++ и перегонялся в С. Первый блин, но он стал не только [комом](#), но и [эльфом](#), и даже [экзешником](#). После долгих доработок, расширений стандарта и попыток сохранить совместимость, научился компилироваться прямо в Assembler и стал намного сложнее оригинала.

- **Lua** — скриптовый язык с португальскими корнями. Прост и приятен, похож на [Python](#) или [JavaScript](#). А ещё встроен как стандарт во многие [MMORPG](#), так что на нём часто пишут [ботов](#). Также встроен в последние версии Медиавики и на нём можно писать модули в [Педивикии](#).
- **vala** — для тех, кто задалбался писать под [GTK](#) на чистом C.
- **Cyclone** — попытка записать все расхожие шаблоны в новые функции и конструкции. Получилось чисто, но использовать всё равно тяжело.
- **Rust** — Cyclone для простых смертных. [Недавно](#) темпы развития были такие, что от коммита к коммиту могли пропадать или меняться ключевые слова.

Эзотерика

Вычисление i-го числа из [ряда Фибоначчи](#) с непредсказуемым поведением программы в итоге. [Из серии](#) «я знаю, что в циклах в C/C++ можно писать всякую [эзотерическую поебень](#)»:

```
int i=8, a1, a2;
for (a1=a2=1; i>2; a1=(a2+=a1)-a1)
/*
    быдлокодеры думают, что это мозгоебалка для школьника,
    а на самом деле a1=a2, а быдло не забывает что такое +=
*/
    i--;
```

Проверка: является ли n степенью 2:

```
!(n & (n-1))
```

Преобразование типа к "bool":

```
!!x
```

Классика: достаём нулевой символ из строки. Нумерация элементов массива начинается с нуля:

```
"abcd"[0]
```

Один из способов проверки C или C++, помимо идентификатора `__cplusplus`. Алсо, многие реализации C до C99 не признают однострочных комментариев `//` языка C++, что может быть использовано для различия двух братских языков:

```
printf("%s",sizeof('C')==sizeof(int)?"C":"C++");
```

Выбор функции:

```
#include <math.h>
```

```
result = (use_cos ? cos : sin)(M_PI);
```

Копирование строк:

```
while( *dst++ = *src++ ) ;
```

[Тут и так все понятно](#):

```
m(f,a,s)char*s;
{char c;return f&1?a!=*s++?m(f,a,s):s[1]:f&2?a!=*s++?1+m(f,a,s):1:f&4?a--?
putchar(*s),m(f,a,s):a:f&8?*s?m(8,32,(c=m(1,*s++,"Arjan Kenter. \no$. ./.\\"),
m(4,m(2,*s++,"POCnWAUvBVxRsoqatKJurgXYyDQbzhLwkNjdMTGeIScHFmpliZef"),&c),s)):
65:(m(8,34,"rgeQjPruaOnDaPeWrAaPnPrCnOrPaPnPjPrCaPrPnPrPaOrvaPndeOrAnOrPnOrP\
n0aPnPjPaOrPnPrPnPrPtPnAaPnBrnnsrnnBaPeOrCnPrOnCaPn0aPnPjPtPnAaPnPrPnPrCaPn\
BrAnxrAnVePrCnBjPrOnvrCnXrAnxrAnsrOnvjPrOnUrOnornnsrnnorOtCnCjPrCtPnCrnnirWtP\
nCjPrCaPnOtPrCnErAn0jPrOnvtPnnrCnNnnRePjPrPtNrUnnrntPnbtPrAaPnCrnnOrPjPrRtPn\
CaPrWtCnKtPnOtPrBnCjPronCaPrVtPnOtOnAtnrxaPnCjPrqnnPrta0rsaPnCtPjPratPnnaPrA\
aPnAaPtPnnaPrvaPnnjPrKtPnWaOrWtOnnaPnWaPrCaPnnt0jPrrtOnWanrOtPnCaPnBtCjPrYtOn\
UaOrPnVjPrwttnnxjPrMnBjPrTnUjP"),0);}

main(){return m(0,75,"mIWltouQJGsBniKYvTx0DAfbUcFzSpMwNCHegrdLaPkyVRjXeqZh");}
```

Да, и это тоже программа. А в C++ и C99 эти примеры работать не будут, ибо нет неявного int:

```
#include <stdio.h>
main (int t, int _, char *a)
{
    return!0<t?t<3?
    main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a)):1,t<_?main(t+1,_,a)
    :3,main(-94,-27+t,a)&&t==2?_<13?main(2,_,1,"%s %d %d\n"):9:16:t<0?t<-72?
    main(_,t,"@n'+,#'/*s{ }w+/w#cdnr/+, { }r/*de}+,/*{*+,/w{ }+,/w#q#n+,/# { }l+,/n\
    {n+,/+n+,/# ;#q#n+,/+k#;*+,/'r : 'd*'3, } {w+K w'K: '+'e#';dq# 'l q#'+d'K#!\
    /+k#;q# 'r }eKK# }w' r }eKK{nl }' /#;#q#n' } { }w' } { } {nl } /+n';d }rw' i;# } {nl }!\
```



```

/n{n#'; r{#w'r nc{nl}']/#{l,+ 'K {rw' iK{;[{nl}]/w#q#n'wk nw' iwk{KK{nl}!/\
w{% 'l##w# ' i; :{nl}']/*{q# 'ld;r'}{nlwb!/*de}'c ;;{nl' -{ }rw']' /+,}## '*' }\
#nc,',#nw]']/+kd'e}+;# 'rdq#w! nr' / ' ) }+}{rl# ' {n' ' )# }'+}##(!/"/)
:t<-50? ==*a?putchar(31[a]):
main(-65,_,a+1):
main((*a=='/')+t,_,a+1):
0<t?main(2,2,"%s")
:*a=='/'||main(0,main(-61,*a,
"!ek;dc i@bK'(q)-[w]*%n+r3#l,{ }:\nuwloca-0;m .vpbks,fxntdCeghiry"
),a+1);
}

```

ОС на один лист? [Лерко!](#) Поддерживает многозадачность, гуй (с мышкой). Имеется загрузчик программ, встроенный интерпретатор команд и текстовый просмотрщик.

```

#define G(n) int n(int t, int q, int d)
#define X(p,t,s) (p>=t&& p<(t+s)&&(p-(t)&1023)<(s&1023))
#define U(m) *((signed char *) (m))
#define F if(!--q){
#define I(s) (int)main-(int)s
#define P(s,c,k) for(h=0; h>>14==0; h+=129)Y(16*c+h/1024+Y(V+36))&128>>(h&7)?U(s+(h&15367))=k:k

```

```

G (B)
{
    Z;
    F D = E (Y (V), C = E (Y (V), Y (t + 4) + 3, 4, 0), 2, 0);
    Y (t + 12) = Y (t + 20) = i;
    Y (t + 24) = 1;
    Y (t + 28) = t;
    Y (t + 16) = 442890;
    Y (t + 28) = d = E (Y (V), s = D * 8 + 1664, 1, 0);
    for (p = 0; j < s; j++, p++)
        U (d + j) = i == D | j < p ? p--, 0 : (n = U (C + 512 + i++)) < ' ' ? p |=
            n * 56 - 497, 0 : n;
}

```

```

n = Y (Y (t + 4)) & 1;
F
U (Y (t + 28) + 1536) |=
62 & -n;
M
U (d + D) =
X (D, Y (t + 12) + 26628, 412162) ? X (D, Y (t + 12) + 27653,
410112) ? 31 : 0 : U (d + D);
for (; j < 12800; j += 8)
    P (d + 27653 + Y (t + 12) + ' ' * (j & ~511) + j % 512,
        U (Y (t + 28) + j / 8 + 64 * Y (t + 20)), 0);
}

```

```

F if (n)
{
    D = Y (t + 28);
    if (d - 10)
        U (++Y (t + 24) + D + 1535) = d;
    else
    {
        for (i = D; i < D + 1600; i++)
            U (i) = U (i + 64);
        Y (t + 24) = 1;
        E (Y (V), i - 127, 3, 0);
    }
}
else
    Y (t + 20) += ((d >> 4) ^ (d >> 5)) - 3;
}
}

```

```

G (_);
G (o);
G (main)
{
    Z, k = K;
    if (!t)
    {
        Y (V) = V + 208 - (I (_));
        L (209, 223) L (168, 0) L (212, 244) _((int) &s, 3, 0);
        for (; 1;)
            R n = Y (V - 12);
        if (C & ' ')
        {

```

```

k++;
k %= 3;
if (k < 2)
{
    Y (j) -= p;
    Y (j) += p += U (&D) * (1 - k * 1025);
    if (k)
        goto y;
}
else
{
    for (C = V - 20;
        !i && D & 1 && n
        && (X (p, Y (n + 12), Y (n + 16)) ? j = n + 12, Y (C + 8) =
            Y (n + 8), Y (n + 8) = Y (V - 12), Y (V - 12) =
                n, 0 : n); C = n, n = Y (n + 8));
        i = D & 1;
        j &= -i;
    }
}
else if (128 & ~D)
{
    E (Y (n), n, 3, U (V + D % 64 + 131) ^ 32);
    n = Y (V - 12);
y:C = 1 << 24;
    M U (C + D) = 125;
    o (n, 0, C);
    P (C + p - 8196, 88, 0);
    M U (Y (0x11028) + D) = U (C + D);
}
}

for (D = 720; D > -3888; D--)
    putchar (D >
        0 ?
            " )!\320\234\360\256\370\256 0\230F      .,mnbvcxz ;lkjhgfdsa \n][poiuytrewq =-0987654321"
            " \357\262 \337\337 \357\272 \337\337 ( )\"343\312F\320!/ !\230 26!/\16 K>!/16\332"
            " \4\16\251\0160\355&\2271\20\2300\355`x{0\355\347\2560 \237qpa%\231o!\230 \337\337\337"
            " , )\"K\240 \343\316qrpxzy\0 sRDh\16\313\212u\343\314qrzy !0( "
            [D] ^ 32 : Y (I (D)));
return 0;
}

G (o)
{
    Z;
    if (t)
    {
        C = Y (t + 12);
        j = Y (t + 16);
        o (Y (t + 8), 0, d);
        M U (d + D) =
            X (D, C, j) ? X (D, C + 1025, j - 2050) ? X (D, C + 2050,
                j - 3075) ? X (D,
                    C + 2050,
                    j -
                    4100) ?
                X (D, C + 4100,
                    ((j & 1023) + 18424)) ? 176 : 24 : 20 : 28 : 0 : U (d + D);
        for (n = Y (t + 4); U (i + n); i++)
            P (d + Y (t + 12) + 5126 + i * 8, U (n + i), 31);
        E (Y (t), t, 2, d);
    }
}

G (_)
{
    Z = Y (V + 24);
    F Y (V - 16) += t;
    D = Y (V - 16) - t;
}

F for (i = 124; i < 135; i++)
    D = D << 3 | Y (t + i) & 7;
}

if (q > 0)
{
    for (; n = U (D + i); i++)

```

```

    if (n - U (t + i))
    {
        D += _(D, 2, 0) + 1023 & ~511;
        i = ~0;
    }
F if (Y (D))
{
    n = _(164, 1, 0);
    Y (n + 8) = Y (V - 12);
    Y (V - 12) = n;
    Y (n + 4) = i = n + 64;
    for (; j < 96; j++)
        Y (i + j) = Y (t + j);
    i = D + 512;
    j = i + Y (i + 32);
    for (; Y (j + 12) != Y (i + 24); j += 40);
    E (Y (n) = Y (j + 16) + i, n, 1, 0);
}
}
}

return D;
}

```

Компилировать gcc-3.4, полученная программа при запуске сгенерирует ядро, которое нужно запустить grub'ом. Ещё нужен fs.tar. [Инструкция по сборке и запуску](#).

См. также

- [Ассемблер](#)
- [Perl](#)
- [C++](#)
- [C Sharp](#)
- [Python](#)
- [Программист](#)
- [Индусский код](#)

Ссылки

- Песня [Write in C](#).
- [эмулируем ООП на чистом C](#)

Примечания

- ↑ ООП закладывалось еще в 60-е, когда его толком и реализовывать-то было не на чем, поэтому его и не замечали.
- ↑ Справедливости ради нужно отметить, что тем же самым вообще страдают языки общего назначения, в которых не реализован механизм автоматической сборки мусора. Ну а особо одарённые [быдлокодеры](#) умудряются устраивать ***memory leaks*** даже в самомусоросборной [Жабе](#) путём формирования так называемых «островов» в памяти — коллекций связанных **между собой** объектов, которые, однако, уже не используются самой программой.
- ↑ Предикат это функция-критерий, например, если взять последние три абзаца, дать их прочитать тебе и посмотреть, поймёшь ты или нет, то текст будет предикатом, ты — аргументом, а результат «[ну ты понел?](#)» позволит отнести тебя к одной из двух групп.



Языки программирования

++i + ++i 1C AJAX BrainFuck C Sharp C++ Dummy mode Erlang Forth FUBAR
 God is real, unless explicitly declared as integer GOTO Haskell Ifconfig Java JavaScript LISP
 My other car Oracle Pascal Perl PHP Prolog Pure C Python RegExp Reverse Engineering
 Ruby SAP SICP Tcl TeX Xyzy Анти-паттерн Ассемблер Быдлокодер
 Выстрелить себе в ногу Грязный хак Дискета ЕГГОГ Индусский код Инжалид дежице
 Капча КОИ-8 Костыль Лог Метод научного тыка Очередь Помолясь Проблема 2000
 Программист Процент эс Рекурсия Свистелки и перделки Спортивное программирование
 СУБД Тестировщик Умение разбираться в чужом коде Фаза Луны Фортран Хакер

w:Си (язык программирования) en:w:C (programming language) ae:C