Kakos_nonos

Язык программирования

СТЕПЛЕР

Полный справочник

Содержание

---

1. Вступление

Можете спокойно пропустить это вступление, потому что его никто и никогда не читает; скажу лишь, что всё упомянутое в этой книге реально существует, и его существование подтверждено научными фактами, а если же что-то из написанного здесь не существует, то я отвечаю за это своей головой.

Но если Вы всё-таки дочитали до сюда, то я расскажу Вам, о чём эта книга. Итак, это книга о языке программирования СТЕПЛЕР, а именно о том, как на нём программировать.

В книге рассмотрены все команды языка, алгоритмические конструкции, (потому, что зная одни лишь команды, ничего не напишешь) и примеры программ. Всё описано простым и доступным языком, поэтому его изучение, несмотря на всю его необычность, покажется вам простым и увлекательным занятием.

Кстати, забыл сказать, что автор не несёт никакой ответственности за то, что Вы там понапишете на этом языке, и вообще, я ни за что не несу ответственности и ни за что не отвечаю. Вот так!

1.1 Происхождение и назначение

Язык степлер был придуман вечером 14 февраля 2011 года мной у себя дома. При создании языка ставились такие цели: сделать минимальный язык программирования с максимальными возможностями, и чтобы программа не была похожа ни на один другой язык. В целом, задумка удалась. В языке всего несколько команд, возможностей много, а код ни похож ни на что другое. Правда, есть некоторое сходство с Befunge, Assembler, но концепции у них разные.

1.2 Название

Название языка расшифровывается как STAck-oriented Programming Language like assemblER (Стек-ориентированный язык программирования типа ассемблера) Как видите, настоящая его аббревиатура это STAPLER, но поскольку это не совпадает с его настоящим названием, то так называть язык запрещено.

1.3 Спасибы

Я хотел бы в первую очередь сказать спасибо себе, за то что сделал этот язык и написал эту книгу. Также хочу выразить благодарность Абадяберу, за то что он написал свой интерпретатор, Admin'у с iForum'а, за то, что сделал этот форум, а также всем остальным, кому за что.

Отдельное спасибо хочу сказать буквам русского алфавита, без чьей помощи эта книга не была бы написана.

2. Основные концепции

В этой части сделана попытка описать, как и почему степлер может быть использован; т.е. что он из себя представляет и для чего хорош.

2.1 Программа-пример

Вот типичная пограмма на степлере, она просит у пользователя ввести массив чисел, сортирует их методом пузырька и выводит их на экран. Как видите, первые 11 строк программы служат для ввода программы, следующие 7 - для сортировки введённого массива, и последние шесть для вывода отсортированного массива. Более подробный анализ этой программы будет дан ниже.

```
[Введите количество элементов]
$(3)(2$)
[1 - По убыванию, 0 - По возрастанию]
$(7)(2$^2*^1-)
$(4)(0)
$(5)(0)
{M}
$(5)(5$^1+)
[Введите элемент]
```

```
$(5$^7+)(2$)
#(5$^3$-|)<M>
{M2}
$(4)(4$^1+)
$(5)(4$^3$^1-%^8+)
$(6)(5$^1+$)
$(5$^1+)(5$$)
$(5$$^6$-^7$*!^1+^2/^5$+)(6$)
#(4$^3$^1-^3$*-|)<M2>
$(5)(0)
[Отсортированный массив]
{M1}
$(5)(5$^1+)
$(2)(5$^7+$)
#(5$^3$-|)<M1>
```

2.2 Области применения

Из-за своего необычного подхода к программированию степлер может использоваться для тренировки мозга и лечения некоторых заболеваний типа старческого слабоумия или атрофии головного мозга. Особенно эффективно чередовать степлер с другими языками типа паскаля или типа того. Это способствует скорейшему нарастанию нервных клеток, в следствие чего может появится суперразум. Не перестарайтесь!

Кроме психушек, степлер может использоваться для моделирования ядерных реакций, вычисления траектории полёта ракет и для многого другого.

Также степлер можно применять для тестирования возможностей программиста, например, при поступлении на работу. Ему можно дать задание и посмотреть, сможет ли он решить его на степлере. Если сможет, то посмотреть, сколько строчек она занимает, и чем меньше их будет, тем лучше.

3. Описание

Здесь будет дано описание самого языка, его команд и операторов. Внимание! Чтение этой и последующих глав начинающими программистами может сильно навредить их психике, будьте осторожны!

3.1 Переменные

В степлере для хранения данных используется одномерный массив. Все переменные хранятся в этом массиве и определяются по номеру ячейки. Есть только один тип данных – 16-битное знаковое целое. То есть, диапазон значений переменной: -32767..32768. Первые две ячейки зарезервированы для ввода/вывода, их нельзя использовать для хранения данных.

3.2 Присваивание

Главной командой в степлере является команда присваивания. С её помощью можно засунуть число в переменную. Эта команда пишется так:

```
$(номер)(значение)
```

Вначале пишется доллар, потом в скобках пишется номер переменной, а затем, тоже в скобках, значение, которое надо запихать в эту переменную. Никакие пробелы не допускаются! Вот пример программы, которая засовывает число 120 в ячейку 3.

```
$(3)(120)
```

Вроде бы всё просто и бессмысленно, да нет. Дело в том, что с помощью этой примитивной команды можно творить очень многое. Да как же? -

да так. Как известно, в бейсике тоже есть команда присваивания, она обозначается символом равно (=). Так вот, с помощью этого присваивания можно делать вычисления вот так: a=2+2*2. Так и в степлере, за число принимается выражение, с помощью которого можно делать вычисления. Это показывает, что степлер ничем не уступает бейсику.

3.3 Операторы

Сейчас мы попробуем описать операторы, которые применяются в числовом выражении. Именно попробуем, потому что это ещё никому и никогда не удавалось.

Итак, начнём. Вначале надо определиться, что это обратная польская запись, использующая стек. Если вы не знакомы с одним из этих понятий, то можете не читать дальше - не поймёте, хотя можете прочитать, исключительно для удовольствия. Перейдём к описанию операторов.

Все операторы пишутся в строку слитно, без пробелов, и выполняются они справа налево без приоритетов. Операторы делятся на несколько групп: базовые, математические, логические, самомодификационные. Всех их можно использовать вместе для получения нужного результата.

И последнее: относитесь вежливо к нашим операторам. Они маленькие, им нужны забота и уход, не обижайте их и не используйте их не по назначению: не делите на ноль и не извлекайте корень из отрицательного числа, и всё будет хорошо.

3.3.1 Базовое

Самый базовый оператор - это числа. Все числа, что встречает на пути интерпретатор,

попадают в стек, при этом цифры по одной дописываются к тому, что уже лежит в стеке. Например, попалось интерпретатору число 165. Он выполняет такие команды: Поскольку вначале на вершине стека лежит ноль, то он дописывает 1 к нулю, и на вершине стека 1; Дописывает к одному 6 и получается 16; Дописывает к 16-ти 5 и плучается 165.

Вот так оно работает.

Ещё один важный оператор - это вверх. Он обозначается **^**. Этот оператор сдвигает на один вверх все значения стека, а вершину делает равной нулю. С помощью этого оператора можно заполнять стек, например, выражение **24^23^41^9** заполнит его так: 9,41,23,24.

Для вызова переменной нужен оператор $ (доллар). Он извлекает из стека число, и засовывает в стек значение переменной(с этим номером). Например, нам надо положить в стек значение восьмой переменной. Мы делаем так: **8$**

3.3.2 Математика

Да, в этой книге будет много математики, поэтому если вы плохо учили её в школе, то рекомендуем взять учебник и подучить, а если неохота, то не читайте дальше.

Степлер поддерживает насколько арифметических действий:

Сложение (**+**)
Вычитание (**-**)
Умножение (*****)
Деление (**/**)
Остаток от деления(**%**)
Корень (**@**)
Степень (**&**)

Все они работают одинаково: берут из стека операнды, выполняют над ними действия и

результат отправляют обратно в стек. Вот несколько примеров:

2^2+	2+2
4^2-	4-2
48^8/	48/8
3^4&	4*4*4
7^2%	Ост. от деления 7 на 2
100@	Корень от 100

Надо кое-что пояснить: деление и корень ЦЕЛОЧИСЛЕННЫЕ! Все операторы удаляют операнды из стека. То есть, до операции сложения, например, стек выглядел так: 2,5,7,5, а после операции - 7,7,5.

Также к арифметическим операциям можно отнести генератор случайных чисел. Он работает так: берёт из стека число и генерирует случайное от нуля до этого числа минус 1. То есть, если на вершине стека 100, то эта команда сгенерирует случайное от 0 до 99.
Эта команда обозначается знаком вопроса (**?**).

3.3.3 Логика

Степлер обладает хорошей логикой (она не подчиняется человеческим правилам и не зависит от настроения), поэтому её можно применять при выборе решений.

Есть несколько логических операторов:

Отрицание (**|**). Если на вершине стека ноль, то он превращается в один, а если же там не ноль, то это значение трансформируется в ноль. Этот оператор очень нужен для логических решений и для модификации чисел.

Знак числа (**!**). Это не совсем логический оператор, но он часто применяется как таковой. Он определяет знак числа, которое лежит на вершине стека. Положительное число

трансформируется в 1, ноль — в ноль, а отрицательное в -1.

3.3.4 Самомодификация

Это очень интересный оператор, он может делать всё, что умеют все вышеописанные операторы вместе взятые. Интересно, правда? Так вот, это оператор самомодификации. Он обозначается кавычками (") и делает вот что: с вершины стека берёт число и выполняет тот оператор, чей код и есть это число. Вот коды всех операторов.

0 - 48
1 - 49

…

8 - 56
9 - 57
$ - 36
^ - 94
+ - 43
- - 45
*** - 42**
/ - 47
@ - 64
& - 38
? - 63
! - 33
| - 124

Оператор самомодификации этим оператором вызван быть не может, потому что будет рекурсия, а это плохо.

Вот пример: строка **2^2^56"** работает так: в стек помещаются числа 2, 2, 56 и выполняется оператор ". Он извлекает из стека число 56 и

складывает следующие по стеку числа 2 и 2, поскольку это код оператора плюс.

3.4 Ввод/Вывод

Ну вот подумайте, решаем мы тут всякие выражения, а результата не видим. Сейчас мы исправим это упущение. Для ввода-вывода у нас есть специальные ячейки памяти: первая и вторая. Первая служит для символьного ввода-вывода, а вторая - для численного. То есть, запись в эти ячейки будет сопровождаться выводом значений на экран, а если читать эти ячейки, то придётся вводить значение с клавиатуры.

Рассмотрим более подробно:

$(2)(177) — При запуске этой программы на экран вылезет число 177.

$(2)(2$) - То, что вы введёте, то и вылезет на экран.

$(2)(1$) - Вы нажимаете кнопку, а на экране код этой кнопки.

Вот, впрочем и весь ввод-вывод, надо только помнить, что первая ячейка служит для символов, а вторая для чисел.

Ах, да, чуть не забыл, что для вывода строк есть простая конструкция:

[Строка, выводимая на экран]

3.5 Условные переходы

Сейчас мы научимся управлять состоянием программы, а именно использовать управляющие конструкции. В степлере есть только одна управляющая конструкция — условный переход. Вот его синтаксис:

#(число)<метка>

Расшифровываю: вначале идёт решётка, потом в круглых скобках число, а потом, между символами больше и меньше, идёт имя метки. Так вот, если то число, которое в скобках, будет равно нулю, тогда производится переход на метку. Метка в программе обозначается в фигурных скобках:

```
{метка}
```

Вот пример:

```
#(5$)<баня>
$(2)(544)
{баня}
```

Эта программа работает так: если значение пятой ячейки равно нулю, то программа переходит по метке баня (идёт в баню), и команда вывода на экран не выполняется, а если же пятая ячейка не равна нулю, то программа продолжает выполнение и выводит на экран число 544.

А как же безусловный переход сделать? А вот так:

```
#(0)<метка>
```

Переход будет производится только тогда, когда значение выражения в скобках равно нулю, а поскольку там всегда ноль, то и всегда переход будет.

3.6 Сказка

Давайте немного отойдём от основной тематики книги и послушаем поучительную сказочку.

Жил да был один человек, которого звали John Fool. Он ничего ни понимал в программировании, и это не доставляло ему никаких неудобств, но в один прекрасный, но несчастливый для него день увидел он программу на C и подумал: «Хм, все

слова английские, все операторы из математики знакомы, дай-ка напишу что-нибудь», и сел писать. Но как он не пытался, его программа не хотела компилироваться, вызывала то одну, то две, то десять ошибок.

Когда же John Fool просидел так за компом десять часов, он резким движением руки разломил монитор на две части, и больше никогда ни прикасался к программированию.

Вот так вот, зная одни лишь команды и операторы, ничего толкового не напишешь. Поэтому далее мы будем описывать алгоритмы степлера.

4. Условия

Если вы читали выше, то в степлере есть переход только по одному условию - по «равно нулю». Но это не значит, что другие переходы невозможно реализовать, их можно сделать с помощью простых комбинаций.

4.1 Неравно

Условие неравно сделать несложно. Нужно просто поставить после условия оператор отрицания (|). Например, вот условный переход по «равно нулю»

```
#(4^3+$^2-)<Метка>
```

А теперь оно же, только по «не равно нулю»:

```
#(4^3+$^2-|)<Метка>
```

Работает это так: оно выполняет условие, и если оно равно нулю, то команда отрицание превращает ноль в единицу, и условный переход (если это условие стоит в переходе) не выполняется. То

есть, если при выполнении получился не ноль, то он становится в ноль.

4.2 Больше, меньше

Условия больше или меньше можно сделать так: Из первого числа отнимаем второе, потом узнаём знак, отнимаем или прибавляем один. Вот, например, сравнение пятой и шестой ячейки и переход на метку, если первое больше.

```
#(5$^6$-!^1-)<метка>
```

Работает это так: из первого числа отнимаем второе, и, если первое больше, то получается положительное число, а если равно или меньше - ноль или отрицательное. Оператором ! определяется знак числа, и получается, что если больше, то 1, если меньше, то -1, а если ноль, то ноль. Потом, если нам надо выполнить условие по «меньше», то надо прибавить один, чтобы из -1 получилось 0, а если нужно условие «больше», то надо отнимать один, чтобы из 1 получилось ноль.

4.3 Больше или равно, меньше или равно

Эти условия сделать очень просто, если знаете, как делать условия больше или меньше. Для них надо сделать так: «больше или равно» это «не меньше», а «меньше или равно» это «не больше». А как делать отрицание, вы знаете, надо в конце поставить |, и всё будет работать.

5. Преобразование чисел

Преобразование чисел, это когда надо какое-то число преобразовать в другое. Например, если в одной переменной число 5, то надо его

превратить в 56, а если какое-то другое число, то в 77.

Это можно сделать тоскливо — не интеллектуальным методом - вот так:

```
#(4$^5-)<M>
$(4)(77)
#(0)<B>
<M>
$(4)(56)
```

Но это противный метод, мы будем делать по-другому.

5.1 Если ноль, то число, иначе - ноль

В таком случае поможет команда отрицания. После вызова числа надо её поставить, и тогда, если это число равно нулю, оно превратится в один, а если не равно нулю - в ноль. Итак, у нас готово преобразование ноль => 1; не ноль => 0

А теперь нам надо умножить получившееся число на то число, которое надо получить, и всё будет готово.

Вот превращение числа из пятой ячейки в 7, если она равна нулю и в ноль, если не так.

```
$(5)(5$|^7*)
```

5.2 Если ноль, то число, иначе - другое число

Делается это на основе предыдущего примера. Надо узнать разность этих чисел и делать предыдущий пример так, как будто должна появиться их разность, а потом прибавить минимальное число.

Вот пример. Если в пятой ячейке число 0, то оно превратится в 17, а если не равно нулю - в 35.

```
$(5)(5$|^18*^17+)
```

Разность этих чисел — 18, мы на это и умножаем, а потом прибавляем меньшее — 17.

5.3 Если больше или меньше, то...

Тут надо вместо вызова числа написать процедуру сравнения, которая была описана выше. Вот например, если число из пятой ячейки больше, чем в шестой, тогда в третью ячейку засунуть 45, а если меньше или равно - то туда засунем 81.

```
$(3)(5$^6$-!^1-|^36*^45+)
```

6. Массивы

В степлере нет стандартной поддержки массивов, поэтому её надо делать самому, и делается это достаточно легко. Вот простейший пример работы с массивом:

```
$(5)(4$$)
```

Тут происходит копирование в пятую ячейку элемента массива, номер которого лежит в четвёртой ячейке. Но тут есть одна загвоздка: если вдруг кто-то захочет прочитать первый элемент массива, то придётся вводить его с клавиатуры, а это не есть гуд, поэтому надо делать массив со смещением, чтобы не задействовались первые две ячейки, и оставалось место для служебных переменных. Вот, например, массив, со смещением в шесть ячеек.

```
$(4$^6+$)(2)
```

Здесь число два записывается в ячейку массива, номер которой лежит в ячейке 4. И при этом первый элемент этого массива будет находится в седьмой ячейке памяти степлера, потому что 6+1=7.

7. Конструкции

В этой главе мы рассмотрим разные конструкции, которые сильно упрощают жизнь программистам. Однако, в степлере их нет, поэтому их надо реализовывать стандартными методами.

7.1 Циклы For

Рассмотрим реализацию цикла for от одного до четырёх. Значение цикла хранится в третьей переменной.

```
$(3)(0)
{for}
$(3)(3^1+)

Тело цикла

#(3$^4-|)<for>
```

Начало отсчёта цикла надо указывать в первой строчке, причём надо указывать на одно меньше, например, если надо начинать с 5, то, если используется третья переменная, надо писать `$(3)(4)`. Последнее значение цикла надо писать в последней строчке, там, где в том примере написано 4.

7.2 Циклы While

Этот цикл работает до тех пор, пока выполняется условие, и если условие неверно, то цикл не выполняется. Его можно реализовать так:

```
{While1}
#(Условие|)<While2>

Тело цикла

#(0)<While1>
```

Тут не пропустите, где написано «условие», после стоит оператор отрицания, будьте внимательны. Метки можно называть, как хотите.

7.3 Циклы Repeat

Цикл Repeat похож на While, только там тело выполняется хотя бы раз. Вот его решение.

```
{repeat}

Тело цикла

#(условие)<repeat>
```

А тут нет отрицания после условия.

7.4 Конструкция IF-THEN-ELSE

Тут тоже всё достаточно просто:

```
#(Условие|)<else>

Если соблюдается условие

#(0)<endif>
```

```
{else}

Если не соблюдается

{endif}
```

Тут тоже после условия палочка стоит.

8. Примеры программ

Здесь мы подробно разберём разные программы на степлере.

8.1 Hello World

Написать хеллоуворлд на степлере «очень сложно»:

```
[Hello World!]
```

Тут просто берётся команда вывода строки на экран и выводит знаменитую строку, но если вы занимаетесь мазохизмом, то можете поступить таким образом:

```
$(1)(72)
$(1)(101)
$(1)(108)
$(1)(108)
$(1)(111)
$(1)(32)
$(1)(119)
$(1)(111)
$(1)(114)
$(1)(108)
$(1)(100)
$(1)(33)
```

8.2 Игра «Угадай число»

Игра «Угадай число» делается на степлере очень просто. Вот она полностью:

```
$(7)(100?^1+)
{ввод}
[Введи число]
$(6)(2$)
#(6$^7$-)<равно>
#(6$^7$-!^1-)<больше>
[введи больше]
#(0)<ввод>
{больше}
[введи меньше]
#(0)<ввод>
{равно}
[молодец!!!]
```

А вот её подробный разбор:

```
$(7)(100?^1+)
```

Загадываем случайное число от 1 до 100. Случайное число помещаем в седьмую ячейку.

```
{ввод}
[Введи число]
$(6)(2$)
```

Вводим число в шестую ячейку с клавиатуры.

```
#(6$^7$-)<равно>
#(6$^7$-!^1-)<больше>
```

Сравниваем введённое число с загаданным. Если равно, идём к метке равно, а если больше, то к метке больше.

```
[введи больше]
#(0)<ввод>
{больше}
[введи меньше]
#(0)<ввод>
{равно}
[молодец!!!]
```

Далее, в зависимости от метки, выводится нужная надпись, и программа отсылается опять к вводу.

8.3 Квадратичное уравнение

Вот программа, решающая уравнение типа $ax^2+bx+c=0$

Она решает только целочисленно, потому что степлер не поддерживает дробных чисел. Вот весь код:

```
[Введите A]
$(5)(2$)
[введите B]
$(6)(2$)
[Введите C]
$(7)(2$)
$(8)(6$^6$*^4^5$^7$**-)
#(8$!^1+)<mn>
[X1=]
$(2)(0^6$-^8$@+^2^5$*/)
[X2=]
$(2)(0^6$-^8$@-^2^5$*/)
#(0)<end>
{mn}
[Дискриминант отрицательный]
{end}
```

А теперь с комментариями.

```
[Введите A]
$(5)(2$)
[введите B]
$(6)(2$)
[Введите C]
$(7)(2$)
```

В переменные 5,6,7 вводятся коэффиценты A,B,C.

```
$(8)(6$^6$*^4^5$^7$**-)
```

В переменную 8 заносится дискриминант.

```
#(8$!^1+)<mn>
```

Если дискриминант меньше нуля, то программа отсылается в конец, пропуская решение.

```
[X1=]
$(2)(0^6$-^8$@+^2^5$*/)
[X2=]
$(2)(0^6$-^8$@-^2^5$*/)
#(0)<end>
```

Вычисляются значения X_1 и X_2, и программа идёт в конец, пропуская сообщение об отрицательности дискриминанта.

```
{mn}
[Дискриминант отрицательный]
{end}
```

Сообщение об отрицательности дискриминанта и конец.

8.4 Числа фибоначчи.

Вот весь текст.

```
[Введите номер]
$(3)(2$^1-)
$(4)(0)
$(5)(0)
$(6)(1)
{C}
#(3$)<E>
$(4)(5$)
$(5)(6$)
$(6)(4$^5$+)
$(3)(3$^1-)
#(0)<C>
{E}
$(2)(6$)
```

А вот разбор. Здесь используется итеративное определение чисел фибоначчи, потому что степлер не функциональный язык.

```
[Введите номер]
$(3)(2$^1-)
```

В ячейку 3 заносим номер числа минус один. Так надо. Почему – не знаю.

```
$(4)(0)
$(5)(0)
$(6)(1)
```

В ячейах 4,5,6 будут храниться первое, второе и третие числа. Вначале третье число должно быть равно одному, остальные – нулям.

```
{C}
#(3$)<E>
```

Начало цикла и условие, по которому, если третья ячейка равна нулю, то идти к выходу.

```
$(4)(5$)
$(5)(6$)
$(6)(4$^5$+)
```

Производим вычисления. В четвёртую ячейку перемещаем значение из пятой ячейки, в пятую из шестой, а в шестую ячейку - сумму пятой и четвёртой ячеек.

```
$(3)(3$^1-)
#(0)<C>
```

Уменьшаем на один значение третьей ячейки (счётчика) и переходим к началу цикла.

```
{E}
$(2)(6$)
```

Конец вычисления и вывод результата на экран.

8.5 Простые числа

Вот полный код программы:

```
[Введите начало]
$(5)(2$)
[Введите конец]
$(6)(2$)
[ ]
{U}
$(7)(0)
$(8)(2)
$(9)(5$^1-)
{D}
```

```
#(5$^8$%|)<N>
$(7)(1)
{N}
$(8)(8$^1+)
#(8$^1-^9$-|)<D>
#(7$^1-)<E>
$(2)(5$)
{E}
$(5)(5$^1+)
#(5$^1-^6$-|)<U>
```

Эта программа работает так: вводятся начало и конец периода и выводятся все простые числа в этом диапазоне. Простые числа вычисляются перебором делителей. В программе есть два вложенных цикла: первый считает от начала промежутка до его конца, а второй считает от числа, на которое указывает первый цикл минус один до двух. А внутри этих циклов есть условие, если число из первого цикла делится на число из вложенного цикла без остатка, тогда переменная семь принимает значение один.

И вот, в конце каждого вложенного цикла происходит проверка: если переменная семь равна одному, значит число на что-то делится и оно не простое, а если не одному, то оно простое и выводится на экран.

8.6 Brainfuck – интерпретатор

Вот вся программа:

```
[Введите программу]
$(5)(5)
{нов}
$(5)(5$^1+)
$(5$)(1$)
$(1)(5$$)
#(5$$^13-|)<нов>
```

```
$(3)(6)
$(4)(500)
$(1)(13)
$(1)(10)
{следком}
#(3$$^43-)<плюс>
#(3$$^45-)<минус>
#(3$$^60-)<пред>
#(3$$^62-)<след>
#(3$$^46-)<вывод>
#(3$$^44-)<ввод>
#(3$$^91-)<начц>
#(3$$^93-)<конц>
#(3$$)<выход>
$(3)(3$^1+)
#(0)<следком>
{плюс}
$(4$)(4$$^1+)
$(3)(3$^1+)
#(0)<следком>
{минус}
$(4$)(4$$^1-)
$(3)(3$^1+)
#(0)<следком>
{пред}
$(4)(4$^1-)
$(3)(3$^1+)
#(0)<следком>
{след}
$(4)(4$^1+)
$(3)(3$^1+)
#(0)<следком>
{вывод}
$(1)(4$$)
$(3)(3$^1+)
#(0)<следком>
{ввод}
$(4$)(1$)
$(1)(4$$)
$(3)(3$^1+)
```

```
#(0)<следком>
{начц}
#(4$$)<а1>
$(3)(3$^1+)
#(0)<следком>
{а1}
$(5)(0)
{а2}
#(3$$^91-|)<а3>
$(5)(5$^1+)
{а3}
#(3$$^93-|)<а4>
$(5)(5$^1-)
{а4}
$(3)(3$^1+)
#(4$)<следком>
#(0)<а2>
{конц}
#(4$$|)<б1>
$(3)(3$^1+)
#(0)<следком>
{б1}
$(5)(0)
{б2}
#(3$$^91-|)<б3>
$(5)(5$^1-)
{б3}
#(3$$^93-|)<б4>
$(5)(5$^1+)
{б4}
$(3)(3$^1-)
#(5$|)<б2>
$(3)(3$^2+)
#(0)<следком>
{выход}
```

Принцип работы этой программы достаточно прост. Вначале вводится программа, потом идёт цикл интерпретации. Там в начале цикла идёт сравнение текущего символа с командами, и, если символ

равен какой-то команде, то программа отсылается для её выполнения на соответствующую метку, а после выполнения она отсылается к началу цикла (метка следком).

8.7 Сортировка пузырьком

Эта программа описывалась в самом начале книги, и она была вам не понятна. Сейчас мы её рассмотрим более внимательно.

```
[Введите количество элементов]
$(3)(2$)
[1 - По убыванию, 0 - По возрастанию]
$(7)(2$^2*^1-)
$(4)(0)
$(5)(0)
{M}
$(5)(5$^1+)
[Введите элемент]
$(5$^7+)(2$)
#(5$^3$-|)<M>
{M2}
$(4)(4$^1+)
$(5)(4$^3$^1-%^8+)
$(6)(5$^1+$)
$(5$^1+)(5$$)
$(5$$^6$-^7$*!^1+^2/^5$+)(6$)
#(4$^3$^1-^3$*-|)<M2>
$(5)(0)
[Отсортированный массив]
{M1}
$(5)(5$^1+)
$(2)(5$^7+$)
#(5$^3$-|)<M1>
```

Программа содержит много всяких интересных приёмов, и её надо разобрать как можно поподробнее.

```
[Введите количество элементов]
$(3)(2$)
```

В ячейку три заносится количество элементов.

```
[1 - По убыванию, 0 - По возрастанию]
$(7)(2$^2*^1-)
```

В ячейку семь заносится условие сортировки, и из значений ноль и один оно сразу преобразовывается в -1 и 1.

```
$(4)(0)
$(5)(0)
{M}
$(5)(5$^1+)
[Введите элемент]
$(5$^7+)(2$)
#(5$^3$-|)<M>
```

Производится ввод в массив значений с клавиатуры. Массив расположен в памяти со смещением в восемь(первый элемент массива – восьмая переменная). Ввод реализован с помощью цикла.

Сейчас начнётся самое интересное- сортировка. Она здесь реализована с помощью метода пузырька. Предположим, количество элементов в массиве – n штук, тогда для сортировки всего массива придётся пройти n раз от начала до предпоследнего элемента массива, сравнивая текущий элемент со следующим.

Это можно сделать с помощью двух вложенных циклов, но это породит больший код и большие проблемы, поэтому мы поступим по-другому. Мы будем использовать один цикл, который считает от одного до n*(n-1). Значение цикла будет храниться в четвёртой ячейке.

```
{M2}
$(4)(4$^1+)
```

Начало цикла. Четвёртая ячейка увеличивается на один.

```
$(5)(4$^3$^1-%^8+)
```

А в пятой ячейке будет находиться указатель на текущую ячейку. Это производится так: берётся остаток от деления значения цикла на (n-1) и прибавляется восемь. Вот таким образом можно избавиться от вложенных циклов.

Теперь надо обменять местами текущую и следующую ячейку, если они не соответствуют условию. Это можно сделать в три этапа.

```
$(6)(5$^1+$)
```

Во временную ячейку (шестую) перемещается значение следующей ячейки.

```
$(5$^1+)(5$$)
```

А в следующую ячейку перемещается значение текущей ячейки. Получается такая ситуация: во временной ячейке значение следующей, а в текущей и следующей – значение текущей.

И вот самая главная команда:

```
$(5$$^6$-^7$*!^1+^2/^5$+)(6$)
```

Эта команда сравнивает значение текущей ячейки и временной, и в зависимости от условия перемещает значение из временной ячейки в текущую или следующую ячейку.

Сейчас мы разберём посимвольно.

```
5$$^6$-
```

Отнимает из текущей ячейки временную. Получается, если текущая меньше временной, то ответ положительный, иначе ноль или положительный.

```
^7$*
```

Вначале мы писали, что можно выбирать тип сортировки: по возрастанию или по убыванию. Так вот, этот тип хранится в седьмой ячейке в виде чисел 1 и -1. Если в этой ячейке число -1, то при умножении на него результата знак поменяет значение на противоположное, и будет казаться, как будто там, где было больше, стало меньше, и наоборот.

```
!^1+^2/
```

Эта комбинация операторов преобразовывает числа -1 и 1 в числа 0 и 1. Получается, если при сравнении первое число больше, то на выходе – 1, а если меньше - 0, а если включена сортировка по другому условию, то всё наоборот.

```
^5$+
```

К полученному числу прибавляем номер текущей ячейки.

И вся эта запись является адресом, в который будет записано значение из временной ячейки. Вот так вот происходит обмен.

```
#(4$^3$^1-^3$*-|)<M2>
```

Конец цикла.

```
$(5)(0)
[Отсортированный массив]
{M1}
```

```
$(5)(5$^1+)
$(2)(5$^7+$)
#(5$^3$-|)<M1>
```

Вывод отсортированного массива. Тут всё просто, поэтому не будем долго зацикливаться.

8.8 Произведение цифр четырёхзначного числа

Эта программа намного короче аналогичных программ на других языках, что показывает преимущество степлера над ними. Вот текст этой программы:

```
$(3)(2$)
$(2)(3$^1000/^3$^1000%^100/^3$^100%^10/^3$^10%***)
```

Вот расшифровка:
Самое интересное представляет вторая строка, в которой производится вычисление.

```
3$^1000/
```

Находится значение четвёртой цифры делением введённого числа на 1000.

```
^3$^1000%^100/
```

Находится значение третьей цифры делением на 100 остатка от деления на 1000.

```
^3$^100%^10/
```

Находится остаток от деления на 100, и делится на 10. Это вторая цифра.

```
^3$^10%
```

Первая цифра находится просто остатком от деления на 10.

```
***
```

Эти числа перемножаются.

8.9 Калькулятор

Этот калькулятор работает благодаря команде самомодификации.

```
$(2)(2$^2$^1$”)
```

Вначале в стек помещаются оба операнда, введённые с клавиатуры, потом помещается знак, тоже введённый с клавиатуры(+,-,*,/), и выполняется команда самомодификации, которая извлекает из стека команду и выполняет её над двумя следующими элементами стека. Вот так вот, видите, какая короткая программа? Это означает, что степлер намного лучше паскаля и ему подобных.

Вот для сравнения аналогичная программа на паскале:

```
Program calc;
Uses crt;
Var x1,x2:integer;
C:char;
Begin
Readln(x1);
Readln(x2);
C:=readkey;
Case c of
‘+’:writeln(x1+x2);
‘-’:writeln(x1-x2);
‘*’:writeln(x1*x2);
‘/’:writeln(x1 div x2);
```

```
End;
Readln;
End.
```

Ну что, замечаете разницу?

10. Интерпретаторы

Вот мы изучили этот язык, и у вас возникло желание написать супер мега программу на степлере. Для этого нам нужен интерпретатор степлера, и у нас есть целых два интерпретатора на ваш выбор.

9.1 Мой интерпретатор

Этот интерпретатор написал я. Он написан на паскале и длинна его кода 226 строк. Его самый большой плюс в том, что он меньше весит, а минус в том, что у него нет сообщений об ошибках.

9.2 Интерпретатор Абадябера

А этот интерпретатор написал товарищ Абадябер. Он тоже написан на паскале, и его размер 526 же строк кода. Его плюс в том, что у него есть сообщения об ошибках.

9.3 Сравнение

В таблице даны сравнительные характеристики интерпретаторов.

	Мой	Абадябера
Построчное выполнение	+	+
Сообщения об ошибках	-	+

Краткость исходного кода	+	-
Цвета	+	-
Красивость исходного кода	-	+
Появился раньше	+	-
Скорость	1.92	1

Как видите, однозначно сказать нельзя, какой лучше, поэтому выбирайте сами.

10. Интересные факты

Степлер был придуман 14 февраля, то есть в день влюблённых, что объясняет чрезмерную любовь к этому языку.

14 февраля – 45-й день в году, а 45 это 9*5, а 9+5=14, что означает номер дня в месяце.

Также дата появления степлера – 734581 день нашей эры, и тут сумма первых трёх цифр равна сумме последних трёх и равна 14.

Оператор переменной в степлере ($) совпадает с обозначением переменной в PHP, что как бы намекает…

Степлер также означает такую бимбочку, с помощью которой можно вставлять скрепки, хотя эти две вещи никак не взаимосвязаны.

Степлер избавляет программиста от многих проблем, например, не надо бояться ошибок несовпадения типов. В степлере просто нет типов данных, поэтому они не могут быть несовместимы.

В степлере не надо объявлять переменные, это делает его на шаг выше таких языков как си или паскаль.

Время, прошедшее от создания INTERCAL до создания степлера равно 330330 часов. Это настолько символично, что просто невозможно

понять, насколько оно символично, и что оно символизирует.

11. Об авторе

Kakos_nonos – получеловек - полуник, популярный в интернете с некоторого времени. Доподлинно неизвестно, существует ли он на самом деле, или это всего лишь интернет-мем, вызванный ошибкой на сервере, хотя некоторые люди и пытаются говорить, что видели его своими глазами и даже общались с ним, но стопроцентно им верить не стоит, потому что нет ни одной подлинной его фотографии, не считая расплывчатых картинок, распространяемых жёлтой прессой.

Но если собрать воедино все достоверные факты, то можно получить его краткую биографию, и она звучит так: Родился Kakos_nonos, вероятно, в 20-ом веке в Солнечной системе, возможно, на планете Земле. В детстве получил сильную психологическую травму, спровоцированную удалением результатов игры в сапёр. Окончил несуществующий институт ненормального программирования. Сейчас – глава несуществующей компании «Кабардинка Компьютерс». Всё, что написано в этой главе, включая эту фразу – фейк.

12. Ссылочки

Мой интерпретатор:
Kabardcomp.narod.ru/kakstep.zip

Интерпретатор Абадябера:
Kabardcomp.narod.ru/abastep.zip

Все программы, описанные здесь:
Kabardcomp.narod.ru/stprogs.zip

Мой сайт:

Kabardcomp.narod.ru

Сайт Абадябера:
Abaduaber.narod.ru

Моё мыло:
Kakos_nonos@mail.ru

КОНЕЦ!