Интерпретируемый язык для вычислений с плавающей точкой IEEE 754

Научный руководитель к.ф.-м.н А. В. Ермоленко
Исполнитель А. Л. Петрунёв

Сыктывкарский государственный университет

2012

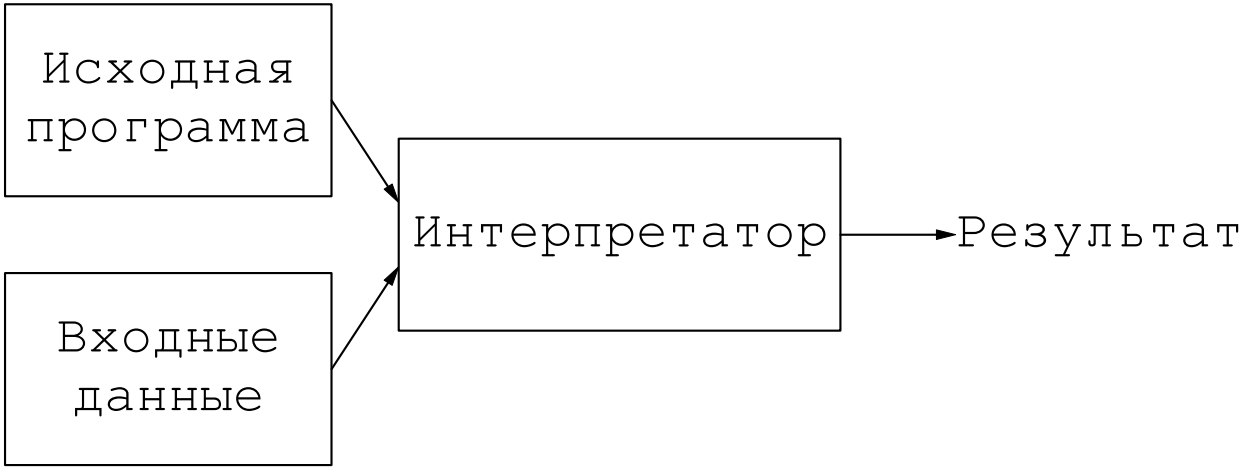
Цель работы

- Создать интерпретатор для языка программирования

1. Обзор

Под **интерпретатором** обычно понимается компьютерная программа, которая выполняет инструкции написанные на языке программирования.

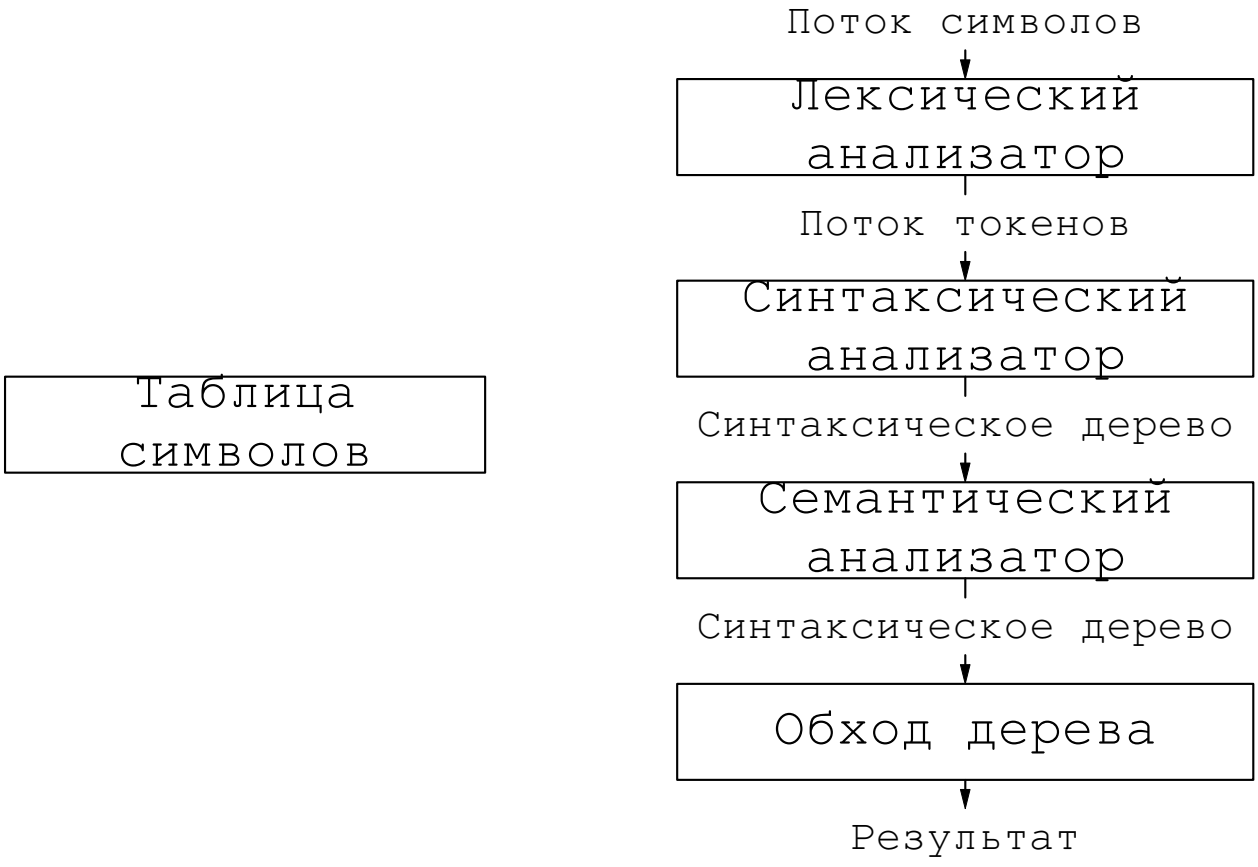
Перед тем как выполнить инструкции, интерпретатор транслирует их в некоторое **промежуточное представление**.


Исходная программа
Входные данные
Интерпретатор
Результат

2. Схема интерпретирования

Процесс интерпретирования можно разделить на несколько последовательных фаз:

- Лексический анализ
- Синтаксический анализ
- Семантический анализ
- Промежуточное представление
- Выполнение


Поток символов
Лексический анализатор
Поток токенов
Синтаксический анализатор
Синтаксическое дерево
Семантический анализатор
Синтаксическое дерево
Обход дерева
Результат
Таблица символов

3. Лексический анализ

Лексический анализатор группирует входные символы в **лексемы**.

Для каждой лексемы создается соотвествующий **токен** в форме:

```
<имя_токена, зачение_атрибута>
```

которая передается в последующую фазу, синтаксическому анализатору.

Например, лексический анализатор нашел число `256.0`.

синтаксический анализатор получит целое число `TOKEN_DOUBLE`, со значением атрибута `256.0`.

Структура хранящая информацию о токене:

```
struct lex {
        token_t         token;
        union {
                char    *id;
                double  real;
                char    *string;
        };
};
```

Предположим у нас есть выражение:

```
y = 0.5*x + b
```

Лексический анализатор видит данную последовательность как:

```
(id, y)(=)(num, 0.5)(*)(id, x)(+)(id, b)
```

где значение атрибута находится после запятой.

4. Синтаксический анализ

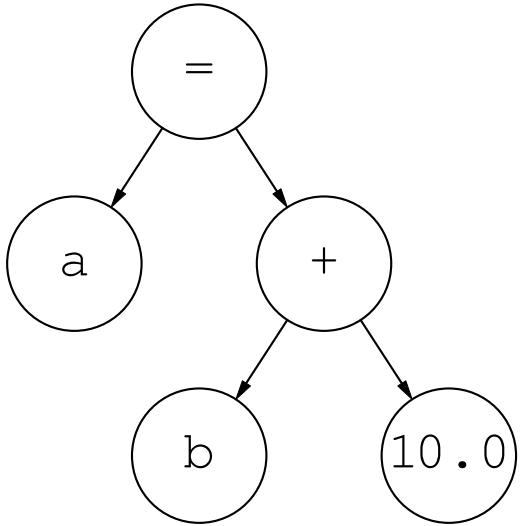
Синтаксический анализатор делает следующее:

- Описывает язык
- Строит синтаксическое дерево, что представляет собой ту же самую программу, только представленную в древовидной форме.

Язык описывается с помощью контекстно-свободных грамматик.

```
summ_expr → mult_expr rest_summ

rest_summ → + mult_expr rest_summ
          | - mult_expr rest_summ | E

mult_expr → term rest_mult

rest_mult → * term rest_mult
          | / term rest_mult | E

term → digit | (summ_expr)

digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

```
a = b + 10.0
```



Структура хранящая информацию об узле дерева для различных операций:

```
struct ast_node_op {
    struct ast_node base;
    struct ast_node *left;
    struct ast_node *right;
    opcode_type_t opcode;
};
```

5. Семантический анализ

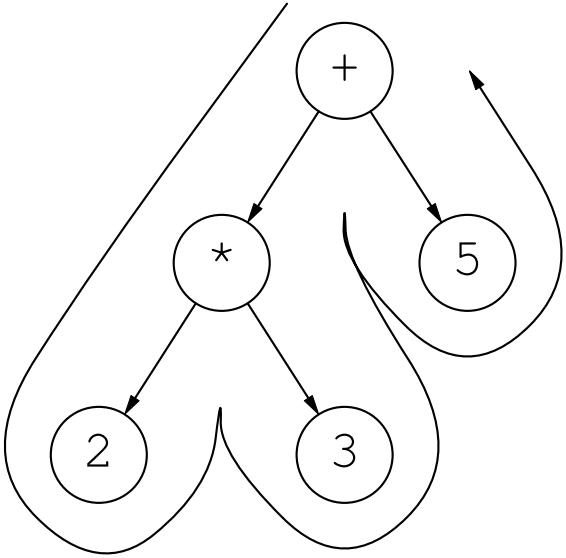
- Семантический анализатор использует синтаксичекое дерево и таблицу символов для проверки семантичекой согласованности с описанием языка.

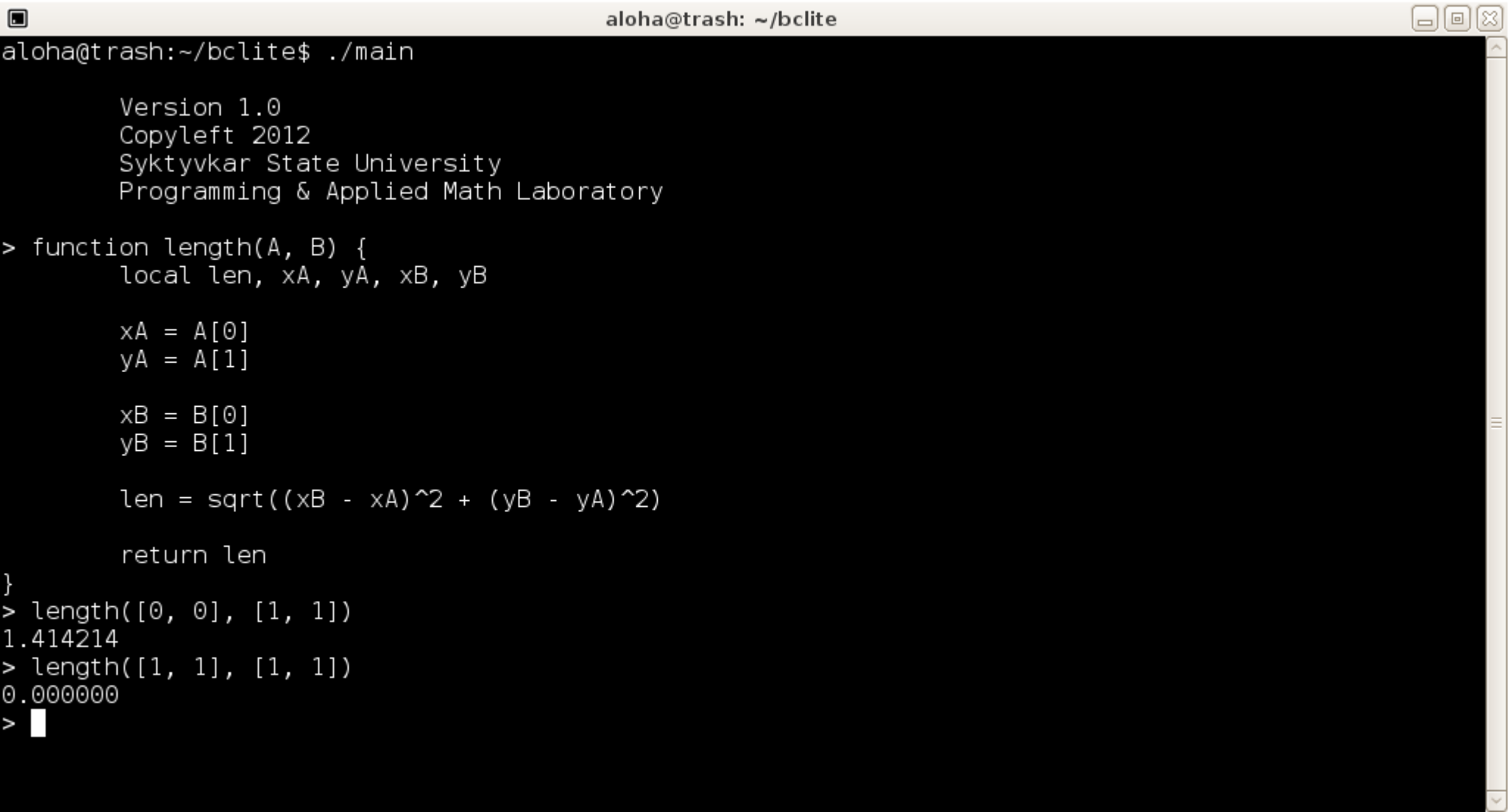
6. Обход дерева

- Обход дерева начинается с **корня**.

- Узлы дерева посещаются рекурсивно, в некотором порядке.

Обход дерева в глубину:




aloha@trash: ~/bclite
aloha@trash:~/bclite$ ./main
Version 1.0
Copyleft 2012
Syktyvkar State University
Programming & Applied Math Laboratory
> function length(A, B) {
local len, xA, yA, xB, yB
xA = A[0]
yA = A[1]
xB = B[0]
yB = B[1]
len = sqrt((xB - xA)^2 + (yB - yA)^2)
return len
}
> length([0, 0], [1, 1])
1.414214
> length([1, 1], [1, 1])
0.000000
>

```
> function factorial(x) {
        local i, res

        i = 1
        res = 1

        if (x == 0.0) {
                return 1
        }

        while (i <= x) {
                res = res*i
                i = i + 1
        }

        return res
}
>
>
>
>
>
> n = 4
> factorial(n)
24.000000
> factorial(0)
1.000000
```


aloha@trash: ~/bclite
>
>
> A = [1, 3; 0, 1]
> b = [3, 1]
> x = b / A
> x
0.000000
1.000000
> A*x
3.000000
1.000000
> b
3.000000
1.000000
> A
1.000000 3.000000
0.000000 1.000000
>
>
>
>
>
>
>
>
>
>
>

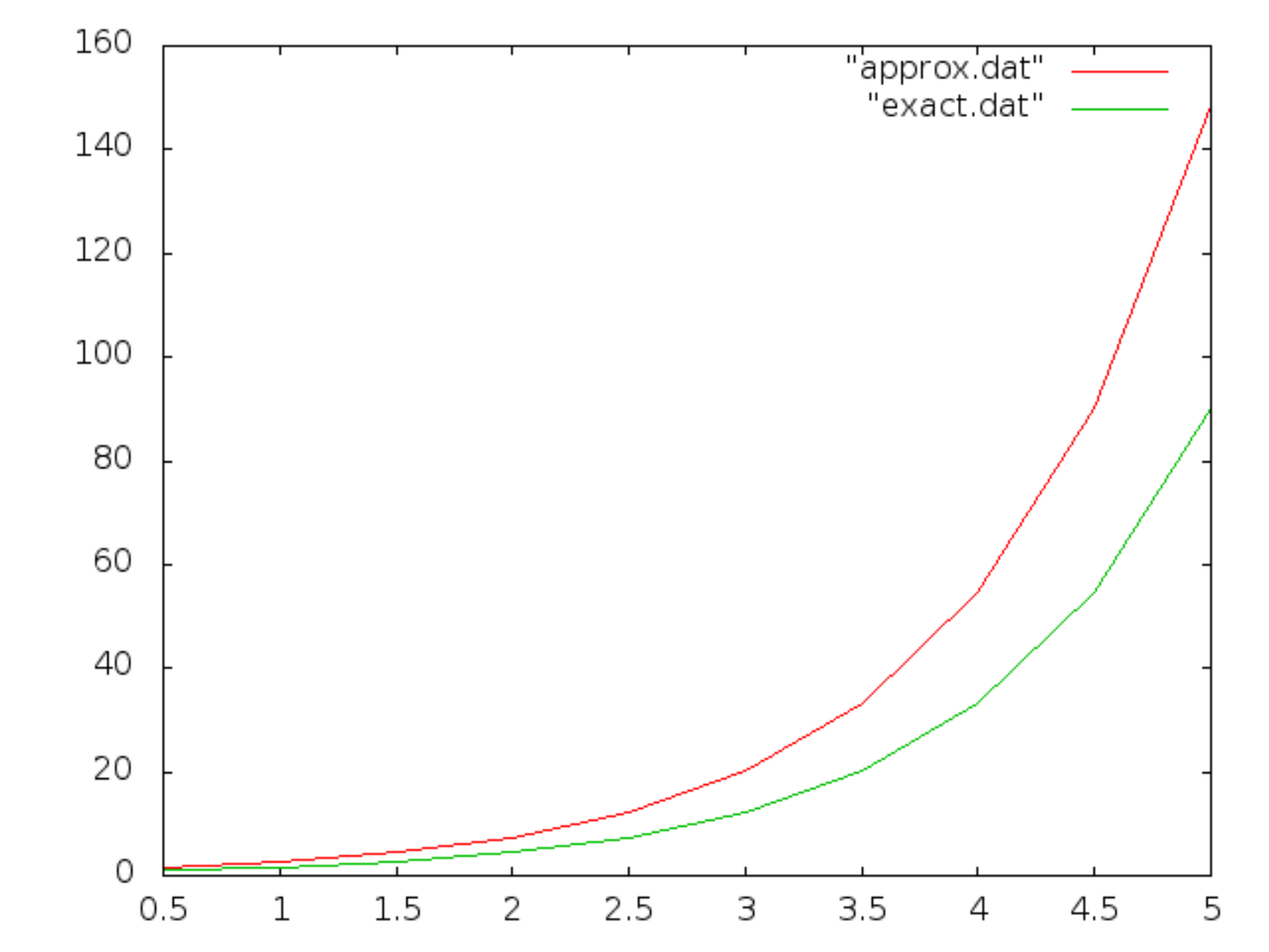
RK4.bc (~/bclite/test) - GVIM3

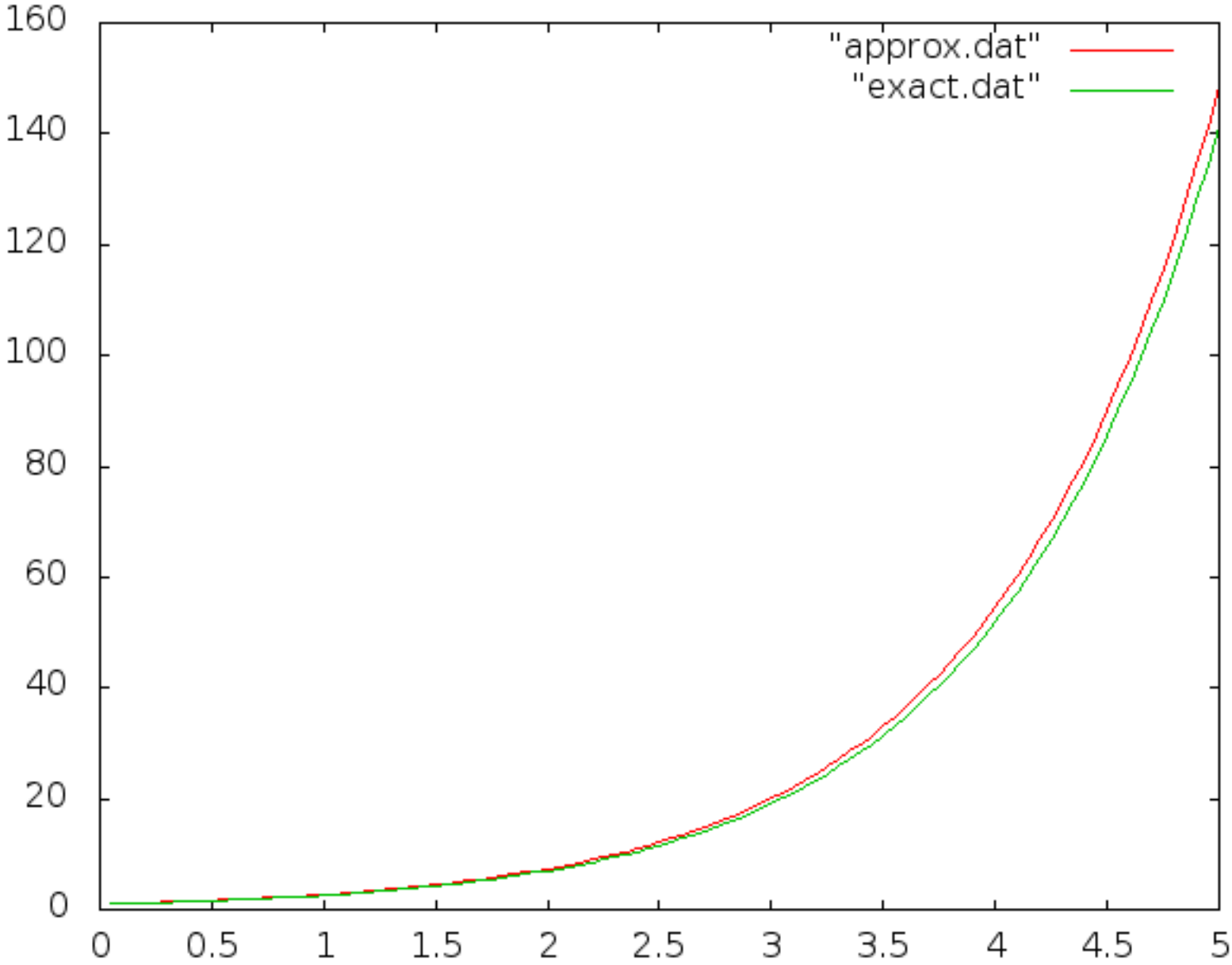
Файл Правка Инструменты Синтаксис Буферы Окно Справка

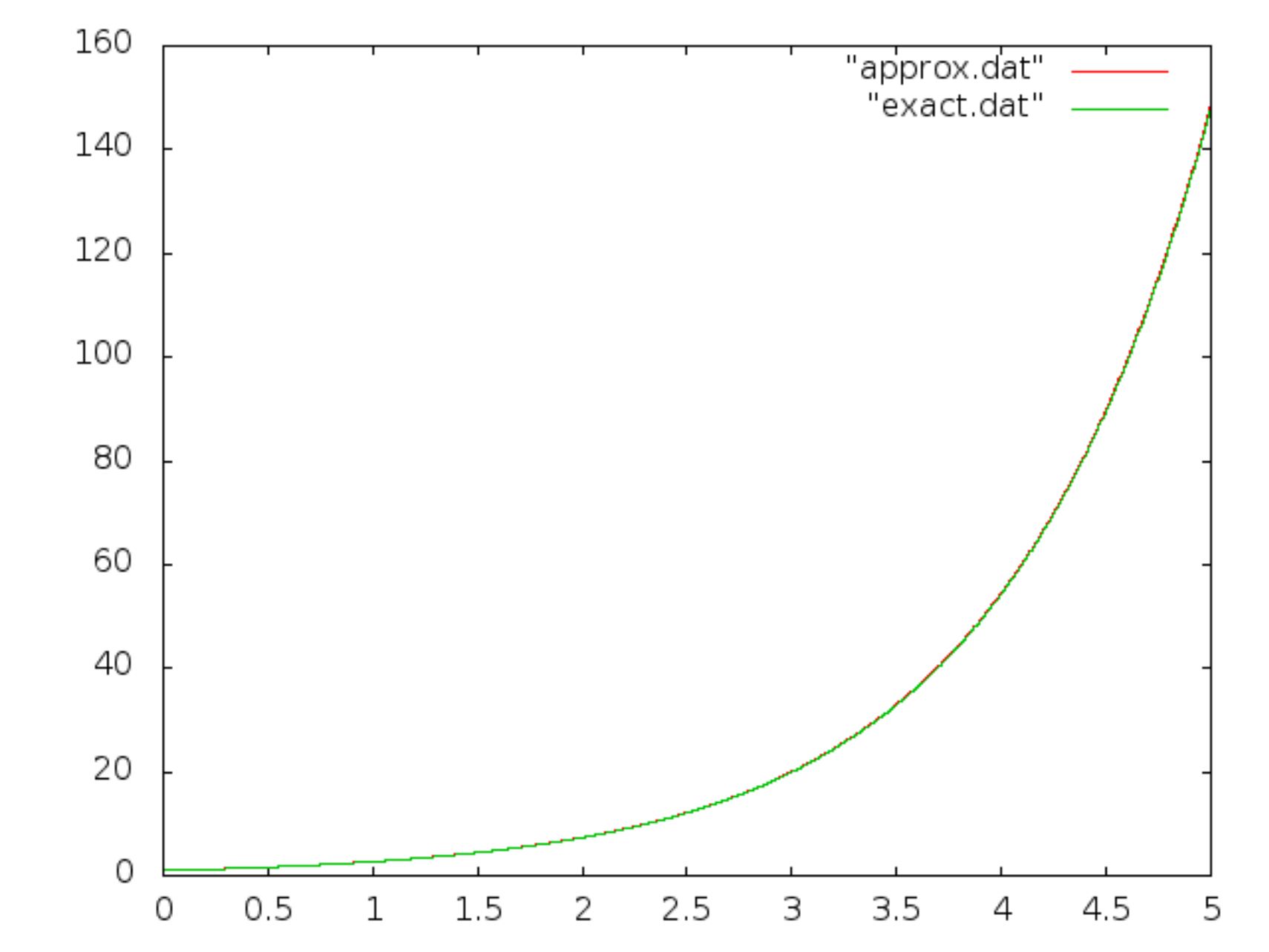
```
15 len = (tn - t0) / h
16 y = vector(len)
17 t = vector(len)
18 dy = vector(len)
19
20
21 for (i = 0; i < len; i = i + 1) {
22         k1 = h*f(t0, y0)
23         k2 = h*f(t0 + 0.5*h, y0 + 0.5*k1)
24         k3 = h*f(t0 + 0.5*h, y0 + 0.5*k2)
25         k4 = h*f(t0 + h, y0 + k3)
26
27         yn = y0 + 1/6*(k1 + 2*k2 + 2*k3 + k4)
28
29         s = solve(t0)
30
31         t0 = t0 + h
32         y0 = yn
33
34         dy[i] = yn
35         y[i]  = s
36         t[i]  = t0
37 }
38
39 "t"
40 t
```

-- ВСТАВКА -- 16,8 63%

aloha@trash: ~/bclite/test
aloha@trash:~/bclite/test$ ../main RK4.bc
t
1.000000
2.000000
3.000000
4.000000
5.000000
_
dy
2.718861
7.391210
20.091972
54.616221
148.462860
_
y
1.000000
2.718282
7.389056
20.085537
54.598150
_
aloha@trash:~/bclite/test$


"approx.dat"
"exact.dat"
160
140
120
100
80
60
40
20
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


"approx.dat"
"exact.dat"
160
140
120
100
80
60
40
20
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


"approx.dat"
"exact.dat"
160
140
120
100
80
60
40
20
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5