Библиотека CalcuLib: компиляция и вычисление выражений

© К. В. Воронцов 18 января 2000 г.

1. Введение

Библиотека предназначена для компиляции и вычисления выражений, записанных традиционным способом. Выражения строятся из констант, переменных, вызовов функций, знаков операций и скобок произвольного уровня вложенности. В целом запись выражений аналогична языкам программирования высокого уровня, таким как C, Basic или Pascal.

Работа с выражениями происходит по следующей общей схеме. Сначала вызывается функция компиляции FC_Compile, которой передаётся строка с описанием вычисляемого выражения. Эта функция преобразует выражение в псевдокод и формирует таблицу переменных. Непосредственно после компиляции все переменные имеют значение empty (пусто). Затем значения переменных должны быть установлены функциями FC_SetVar. Для вычисления значения выражения вызываются функции FC_Calc.

1.1. Инициализация и завершение

Перед началом работы библиотеку необходимо инициализировать вызовом функции FC_Start, а по окончании работы закрыть функцией FC_Finish. После FC_Finish можно вызывать только функцию FC_GetMemoryReport, которая выдаёт отчёт об использовании памяти.

1.2. Таблица выражений

Библиотека позволяет работать с несколькими выражениями одновременно. Все активные в данный момент времени выражения сведены в одну таблицу и пронумерованы, начиная с 0. Доступ к выражениям осуществляется по их порядковым номерам — идентификаторам ExprID. Пользователь библиотеки может либо взять распределение идентификаторов на себя, либо использовать функцию FC_GetFreeExprID для получения наименьшего незанятого идентификатора.

1.3. Функции возвращают код ошибки

Все функции возвращают целое число. Если оно отрицательно, это код ошибки:

(–1) библиотека не инициализирована функцией FC_Start;

(–2) неверный идентификатор выражения ExprID;

(–3) неверный идентификатор переменной VarID.

Неотрицательные значения свидетельствуют об отсутствии ошибки. Функции FC_GetFreeExprID и FC_GetVarID возвращают идентификаторы, неотрицательные в случае отсутствия ошибок. Все остальные функции возвращают 0 в случае отсутствия ошибок.

1.4. Основные типы данных

Выражения и переменные могут иметь различные типы, но все они приводятся к трём основным типам int, real и string. Поэтому в библиотеке имеется по три версии для каждой функции, работающей со значениями переменных и выражений:

- функции FC_CalcNumber, FC_CalcInteger и FC_CalcString вычисляют значение выражения;
- функции FC_SetVarNumber, FC_SetVarInteger и FC_SetVarString задают значения переменных;
- функции FC_GetVarNumber, FC_GetVarInteger и FC_GetVarString выдают значения переменных.

1.5. Передача строк в функции и из функций библиотеки

Строковой тип данных, используемый при передаче строк в библиотеку и из библиотеки, определён как

```
typedef const char* FC_String;
```

Главное правило при работе со строками типа FC_String:

- если строка передаётся в функцию библиотеки в качестве входного аргумента как FC_String StrArgument, то эта строка формируется и освобождается вне библиотеки, и функции библиотеки не имеют права её модифицировать;
- если указатель на строку передаётся в функцию библиотеки в качестве выходного аргумента как FC_String* StrArgument, то эта строка формируется и освобождается внутри библиотеки, и пользовательское приложение не имеет права её модифицировать.

2. Функции библиотеки CalcuLib

2.1. Общие функции

```
int FC_Start ()
```

Инициализация библиотеки. Возвращает 0, если инициализация прошла успешно и 1 при попытке повторной инициализации.

```
int FC_Finish ()
```

Завершение работы. Возвращает 0, если библиотека освобождена успешно и 1, если освобождение не может быть выполнено. Количество вызовов функции FC_Finish должно в точности равняться количеству вызовов FC_Start.

```
int FC_GetFreeExprID ()
```

Получить наименьший незанятый идентификатор выражения. В случае успеха возвращает неотрицательное число, которое может передаваться другим функциям библиотеки в качестве аргумента ExprID.

```
int FC_GetLastMessage (FC_String* Message)
```

Выдать последнее сообщение об ошибке, обнаруженной при компиляции или вычислении выражения. Вызов любой из функций FC_Compile и FC_Calc затирает старое сообщение. Если компиляция или вычисление выполнились без ошибки, функция возвращает 1 и пустое сообщение Message. Если ошибка была, функция возвращает 0 и устанавливает указатель Message на строку сообщения.

```
int FC_GetMemoryReport (FC_String* Buffer)
```

Выдать отчёт о текущем использовании памяти. Функцию можно вызывать как до, так и после вызова FC_Finish. Отчёт выдаётся в следующей форме:

```
CalcuLib memory report:
Number of active allocations: 0 after 260
Allocated size: 0 bytes, maximum 17 Kbytes
Allocated DynArray: 0
Allocated String: 0
Allocated Datum: 0
```

2.2. Компиляция выражений

```
int FC_Compile (int ExprID, FC_String Source)
```

Скомпилировать выражение, заданное строкой Source, записав его в таблицу выражений под номером ExprID. Функция FC_Compile уничтожает старое выражение, хранящееся под номером ExprID и заменяет его новым, заданным строкой Source. Функция возвращает (–5), если при компиляции обнаружена ошибка. Текст сообщения об ошибке выдаётся функцией FC_GetLastMessage.

```
int FC_GetSource (int ExprID, FC_String* Source)
```

Выдать исходную строку выражения.

```
int FC_GetPseudocodeListing (int ExprID, FC_String*
    Listing)
```

Выдать листинг псевдокода. Функция используется в отладочных целях.

2.3. Вычисление скомпилированного выражения

```
int FC_CalcNumber (int ExprID, double* Result)
```

Вычислить значение выражения. Результат преобразуется к вещественному типу и записывается по адресу Result. Если результат равен empty или при вычислении обнаружена ошибка, функция записывает в Result нуль и возвращает (–6). Текст сообщения об ошибке выдаётся функцией FC_GetLastMessage.

```
int FC_CalcInteger (int ExprID, int* Result)
```

Вычислить значение выражения. Результат преобразуется к целому типу и записывается по адресу Result. Если результат равен empty или при вычислении обнаружена ошибка, функция записывает в Result нуль и возвращает (–6). Текст сообщения об ошибке выдаётся функцией FC_GetLastMessage.

```
int FC_CalcString (int ExprID, FC_String* Result)
```

Вычислить значение выражения. Результат преобразуется к строковому типу и записывается по адресу Result. Если результат равен empty или при вычислении обнаружена ошибка, функция записывает в Result указатель на пустую строку и возвращает (–6). Сообщение об ошибке выдаётся функцией FC_GetLastMessage.

2.4. Работа с таблицей переменных

```
int FC_GetTableSize (int ExprID)
```

Выдать размер таблицы переменных для заданного выражения ExprID.

```
int FC_GetVarID (int ExprID, FC_String VarName)
```

Найти по имени переменной её номер в таблице. Возвращает идентификатор переменной, который в дальнейшем может передаваться функциям доступа к переменным в качестве аргумента VarID. Возвращает код ошибки (–4), если переменная с указанным именем отсутствует в данном выражении.

```
int FC_GetVarName (int ExprID, int VarID, FC_String*
    VarName)
```

Выдать имя переменной с порядковым номером VarID в таблице переменных выражения ExprID.

2.5. Запись и считывание значений переменных

```
int FC_SetVarNumber (int ExprID, int VarID, double Value)
```

Задать значение вещественной переменной VarID в выражении ExprID.

```
int FC_SetVarInteger (int ExprID, int VarID, int Value)
```

Задать значение целочисленной переменной VarID в выражении ExprID.

```
int FC_SetVarString (int ExprID, int VarID, FC_String
    Value)
```

Задать значение строковой переменной VarID в выражении ExprID.

```
int FC_GetVarNumber (int ExprID, int VarID, double *Value)
```

Выдать значение переменной VarID в выражении ExprID, преобразовав его к вещественному типу. Если значение переменной пусто (равно empty), функция записывает в Value нуль и возвращает код ошибки (-7).

```
int FC_GetVarInteger (int ExprID, int VarID, int *Value)
```

Выдать значение переменной VarID в выражении ExprID, преобразовав его к целочисленному типу. Если значение переменной пусто (равно empty), функция записывает в Value нуль и возвращает код ошибки (-7).

```
int FC_GetVarString (int ExprID, int VarID, FC_String
    *Value)
```

Взять значение переменной VarID в выражении ExprID, преобразовав его к строковому типу. Если значение переменной пусто (равно empty), функция записывает в Value пустую строку и возвращает код ошибки (-7).

3. Синтаксис и семантика выражений

Выражения образуются из констант, переменных, функций, знаков операций и скобок произвольного уровня вложенности. Правила построения выражений практически те же, что и в языках высокого уровня, таких как Basic, Pascal или C.

Небольшое отличие имеется в операциях сравнения. В отличие от обычных языков программирования запись 0 <= x < 1 корректна и является сокращением более длинного выражения 0 <= x and x < 1, то есть имеет обычный математический смысл. Допускаются цепочки сравнений произвольной длины, включающие любые из 6 операций сравнения, например 0 < x <= y1 = y2 > 1.

Имена переменных могут включать любые символы. Если имя содержит спецсимволы или пробелы, его следует заключить в апострофы. Если имя само содержит апостроф, перед этим апострофом необходимо поставить символ «\». Имена переменных, функций и предопределённых констант не чувствительны к регистру, например sin, SIN и sIn — три корректных имени функции синуса.

3.1. Типы данных

Библиотека оперирует значениями следующих типов:

- логический bool (0 или 1);
- целый int (4-байтовые целые числа);
- вещественный real (8-байтовые вещественные с двойной точностью);
- строковой string (строки произвольной длины);
- дата-время datetime (число секунд, прошедших с 1 января 1970);
- ошибка error.

Приведение типов в выражениях осуществляется автоматически, но можно использовать также функции преобразования типа (см. ниже). Типы bool и datetime являются подмножествами int. При приведении числовых типов к bool положительные значения преобразуются в true, отрицательные и 0 — в false. Тип error эквивалентен строковому типу и содержит сообщение об ошибке.

3.2. Константы

Целочисленные константы можно задавать в обычном виде, а также в бинарной и шестнадцатеричной системах счисления. В двоичных и шестнадцатеричных константах можно использовать точку для визуального отделения разрядов, при вычислениях эта точка игнорируется, например:

0b0000.1000 — двоичное 8
0xFFFF — шестнадцатеричное 65535

Вещественные константы задаются обычным образом — либо в экспоненциальной форме, либо с плавающей точкой: 20.345, 2.345e+8, 345E-8.

Строковые константы записываются в двойных кавычках.

Константы даты-времени задаются как строковые с преобразованием типа: datetime("1997-8-21 11:15:20"). В датах используются разделители "-" или "/".

Неинициализированные переменные имеют значение empty. Преобразование empty к числовым типам даёт 0, к строковому типу — "empty".

По умолчанию в библиотеке предопределены значения некоторых констант, которые можно использовать во всех выражениях:

```
empty;
false      = 0;
true       = 1;
pi         = 3.1415926535897931;
SecMinute  = 60;
SecHour    = 60 * 60;
SecDay     = 60 * 60 * 24;
SecWeek    = 60 * 60 * 24 * 7;
```

3.3. Строковые константы со вставками

Внутрь строковой константы можно вставить произвольное выражение, окружив его квадратными скобками. Строки со вставками используются для сокращения записи конкатенаций. Например, следующие два выражения эквивалентны, но второе имеет более простую запись:

"Число пи=" & PI & ", более точно пи=" & format ("%.16f", PI) & "."
"Число пи=[PI], более точно пи=[PI %.16f]."

Результатом обоих выражений будет строка

```
Число пи=3.14159, более точно пи=3.1415926535897931.
```

После выражения-вставки может следовать формат вывода, который отделяется от него знаком процента и продолжается до закрывающей квадратной скобки. Такие же форматы вывода используются в функции format.

Числовые форматы задаются по правилам функции printf языка C. Числовой формат может содержать только один знак процента, обязан завершаться одной из ключевых букв d, i, o, u, x, X, f, e, E, g, G, и может включать в себя следующие необязательные параметры:

[*flag*] — комбинация флагов –, +, 0, пробел, #;
[*width*] — минимальная длина выводимой строки в символах;
[**.***prec*] — число знаков после запятой при выводе вещественных чисел.

Спецификация	**Назначение**
%[*flag*][*width*]**d** или **i**	целое число со знаком
%[*flag*][*width*]**u**	целое число без знака
%[*flag*][*width*]**o**	восьмеричное целое число без знака
%[*flag*][*width*]**x**	шестнадцатеричное целое без знака, используются прописные буквы a, b, c, d, e, f
%[*flag*][*width*]**X**	шестнадцатеричное целое без знака, используются заглавные буквы A, B, C, D, E, F
%[*flag*][*width*][**.***prec*]**e** **%**[*flag*][*width*][**.***prec*]**E**	вещественное в экспоненциальной форме: 1.2e4, 3.1E-4
%[*flag*][*width*][**.***prec*]**f**	вещественное с плавающей точкой: 12000.0, 0.00031

%[*flag*][*width*][**.***prec*]**g** **%**[*flag*][*width*][**.***prec*]**G**	кратчайшая из двух форм **e** и **f**

Для дат и времени формат вывода задаётся по правилам функции strftime языка C. В отличие от предыдущего случая любое количество форматов можно объединить для вывода одного значения. Например, результатом выражения

"Today [now() %#d %B %Y, %A, %H:%M:%S]."

будет строка вида

```
Today 4 February 2000, Monday, 15:04:55.
```

Необязательный аргумент [*flag*] может принимать только одно значение #. Для форматов %#c и %#x он выводит дату в более подробной форме. Для остальных форматов — удаляет лидирующие нули (4 вместо 04).

Спецификация	**Назначение**
%[*flag*]**c**	стандартное представление даты и времени
%[*flag*]**x**	стандартное представление даты
%X	стандартное представление времени
%[*flag*]**Y**	четырёхзначный год (от 00 до 99)
%[*flag*]**y**	последние две цифры года (от 00 до 99)
%B	полное название месяца
%b	сокращённое название месяца
%[*flag*]**m**	номер месяца (от 01 до 12)
%[*flag*]**d**	день месяца (от 01 до 31)
%[*flag*]**H**	часы (от 00 до 24)
%[*flag*]**I**	часы (от 00 до 12)
%p	индикатор до полудня / после полудня
%[*flag*]**M**	минуты (от 00 до 59)
%[*flag*]**S**	секунды (от 00 до 59)
%a	сокращённое название дня недели
%A	полное название дня недели
%[*flag*]**w**	порядковый номер дня недели (от 0 до 7, 0=воскресенье)
%[*flag*]**j**	порядковый номер дня в году (от 001 до 366)
%[*flag*]**U**	порядковый номер недели в году (от 00 до 51), если неделя начинается с воскресенья
%[*flag*]**W**	порядковый номер недели в году (от 00 до 51), если неделя начинается с понедельника
%z **%Z**	полное и сокращённое название часового пояса

Некоторые форматы даты-времени (d, x, X) конфликтуют с числовыми форматами. Если одна из этих букв оказалась последней в формате даты-времени, после неё следует поставить букву T:

"Today [now() %#xT]."

3.4. Приоритет операций

В выражениях допускаются следующие операции (в порядке приоритета):

,	операция последовательного вычисления (запятая)
:=	операция присваивания
or	логическое ИЛИ
and	логическое И
not	логическое НЕ
= <= >= <> < >	6 операций сравнения
&	операция конкатенации строк
+ –	операции арифметического сложения и вычитания
* /	операции арифметического умножения и деления
^	операция возведения в степень

Приоритет операций может быть изменён с помощью скобок.

Операции присваивания и последовательного вычисления используются для сокращения записи и повышения скорости вычислений. Например, следующие два выражения выдают один и тот же результат, но второе вычисляется эффективней:

```
exp(23.6780*x*x+1.2905*x-45.892) + 1/exp(23.6780*x*x+1.2905*x-45.892)
T:=exp(23.6780*x*x+1.2905*x-45.892), T+1/T
```

Результатом операции присваивания является значение выражения в правой части от знака :=. Результатом операции последовательного вычисления является последнее выражение в списке. Смысл этих операций и их использование полностью аналогичны языку C.

Переменные, стоящие в левой части присваивания являются локальными переменными данного выражения и не заносятся в таблицу переменных. Их значения нельзя устанавливать и считывать с помощью функций FC_SetVar и FC_GetVar.

3.5. Функции

Функции в общем случае могут иметь произвольное число аргументов, которые записываются после имени функции в скобках через запятую. По числу аргументов функции разбиваются на 4 группы:

- Функции с фиксированным числом аргументов (sqrt, sin, exp, mod).
- Функции с необязательными аргументами (log, rand, setbit).
- Функции с произвольным числом аргументов (min, max, if, steps, xor).
- Функции без аргументов (now(), rand()), пустые скобки обязательны.

Ниже приводится полный перечень допустимых функций с их спецификациями и описаниями. Спецификации указывают типы всех аргументов и возвращаемого значения.

При записи спецификаций необязательные аргументы заключаются в квадратные скобки [ ]. Аргументы, повторяемые любое число раз (в том числе ни разу), заключаются в фигурные скобки { }. Слово any означает, что аргумент может иметь произвольный тип.

Функции преобразования типов

bool	bool (any)	преобразование any в логический тип
int	int (any)	преобразование any в целый тип
real	real (any)	преобразование any в вещественный тип
string	string (any)	преобразование any в строковой тип
datetime	datetime (any)	преобразование any в дату и время

Функции проверки значений

IsEmpty	bool (any)	TRUE если any пустое или ошибочное
IsNumber	bool (any)	TRUE если any приводимо к типу real
IsTime	bool (any)	TRUE если any приводимо к типу datetime

Числовые функции с фиксированным числом аргументов

mod	real (real1, real2)	остаток от деления real1 íà real2
abs	real (real)	абсолютная величина (модуль) числа real
sign	int (real)	сигнум числа real, принимает значения –1, 0, 1
max	real ({real})	максимум чисел
min	real ({real})	минимум чисел
sqrt	real (real)	квадратный корень числа real
sin	real (real)	синус числа real
cos	real (real)	косинус числа real
tg	real (real)	тангенс числа real
arcsin	real (real)	арксинус числа real
arccos	real (real)	арккосинус числа real
arctg	real (real)	арктангенс числа real
sh	real (real)	гиперболический синус числа real
ch	real (real)	гиперболический косинус числа real
th	real (real)	гиперболический тангенс числа real
exp	real (real)	экспонента числа real
log	real (real1,[real2])	логарифм real1 по основанию real2, либо натуральный, если real2 опущен

rand	real ([real1],[real2])	случайное число из диапазона [real1,real2), [0,real1) или [0,1)
round	real (real)	округление до ближайшего целого
uproround	real (real)	ближайшее большее или равное целое (потолок)
dnround	real (real)	ближайшее меньшее или равное целое (пол)
MinDiv	int (int)	наименьший делитель int (сам int если простое)
dnSimple	int (int)	наибольшее простое число, не превышающее int

Логические функции

xor	bool ({bool})	логическое исключающее ИЛИ

Битовые функции для целых чисел

Bit	bool (int1,int2)	int2-ый бит в целом числе int1
SetBit	int (int1, [int2],[bool])	установить int1-ый бит числа int2 (0 если опущен) в bool (1 если опущен)
NotBit	int (int)	побитовое отрицание
OrBit	int ({int})	побитовое ИЛИ
AndBit	int ({int})	побитовое И
XorBit	int ({int})	побитовое исключающее ИЛИ

Функции для работы с датой и временем

now	datetime ()	текущая дата-время
time	datetime (any)	преобразование any в datetime с выделением времени
date	datetime (any)	преобразование any в datetime с выделением даты
year	int (datetime)	выделение года из datetime
month	int (datetime)	номер месяца в году, нумерация с 1
day	int (datetime)	номер для в месяце, нумерация с 1
hours	int (datetime)	выделение часов с полуночи из datetime
minutes	int (datetime)	выделение минут из datetime
seconds	int (datetime)	выделение секунд из datetime
WeekDay	int (datetime)	номер дня недели, ВС=0, ПН=1, ..., СБ=6
YearDay	int (datetime)	номер дня в году

Функции для работы со строками

len	int (string)	длина строки

pos	string (string1, string2)	позиция строки string2 в строке string1, или (–1) если не найдена; нумерация с 0
left	string (string,int)	головная часть строки длиной int
right	string (string,int)	хвост строки длиной int
mid	string (string, int1,int2)	средняя часть строки начиная с позиции int1 длиной int2, нумерация с 0
format	string (string, any)	форматный вывод any согласно спецификации, заданной в string

Условные и пороговые функции

if	any ({bool1, any1}, [anyLast])	условное выражение, возвращает: any1, если bool1=TRUE, any2, если bool2=TRUE, и так далее..., anyLast, если все нечётные аргументы равны FALSE.
steps	int (real1, {real})	ступенчатая функция, возвращает: 0, если real1<real2, 1, если real2<=real1<real3, 2, если real3<=real1<real4, и так далее.
switch	any (any, {any1,any2}, [anyLast]}	переключатель значений, возвращает: any2, если any=any1, any4, если any=any3, и так далее... anyLast, если ни одно равенство не выполнено.

4. Тестовая программа TestCalc

Консольное приложение TestCalc предназначено для тестирования библиотеки CalcLib. При обнаружении неправильно обрабатываемых выражений выходной текст, формируемый этой программой, следует передать разработчику.

Программа считывает строки со стандартного ввода, передаёт их функции CL_Compile и записывает результаты в стандартный вывод. Пустые строки и строки, начинающиеся символом “#”, игнорируются.

После компиляции очередной строки в выходной поток выдаётся само выражение, сообщение об ошибке компиляции (если есть) и листинг псевдокода. Выдаётся также список переменных с их значениями (которые должны быть равны empty), сообщение об ошибке вычисления (если есть) и значение вычисленного выражения.

Строки, начинающиеся символом “;”, воспринимаются как список переменных и их значений для последнего скомпилированного выражения. Пробелы вокруг символов “;” и “=” недопустимы. После обработки каждой такой строки снова выдаётся список переменных с их значениями, сообщение об ошибке вычисления (если есть) и значение вычисленного выражения при заданных значениях переменных.

При запуске программы TestCalc с опцией –s в выходной поток выдаётся только выражение и его значение.

Пример входного файла:

```
3*(5+3-x*value)-sqrt(9)
;x=-0.5;value=4;
sqrt(56)
```

По окончании чтения входного потока выдаётся отчёт об использовании памяти, в котором все значения, кроме общего числа выделений и максимального потребовавшегося объёма памяти, должны быть равны нулю.

Ниже в качестве примера использования функций библиотеки CalcuLib приводится несколько упрощённый текст программы TestCalc на языке C.

```
#include <stdio.h>
#include <string.h>
#include "CalcuLib.h"

// компиляция выражения, выдача листинга и сообщения об ошибке
void compile (int ExprID, const char* expression) {
  FC_String str;
  FC_Compile (ExprID, expression);   // компиляция
  FC_GetSource (ExprID, &str);       // получение исходной строки
  printf ("SOURCE %s", str);         // печать исходной строки
  if (FC_GetLastMessage (&str)==0)   // сообщ. об ошибке компиляции
    printf ("COMPILATION ERROR %s\n", str);
  FC_GetPseudocodeListing (ExprID, &str);
  printf ("PSEUDOCODE\n%s", str);    // листинг псевдокода
}
```

```
// вычисление выражения, выдача переменных и сообщения об ошибке
void calculate (int ExprID) {
  FC_String varname, str;
  // перебор всех переменных из таблицы переменных
  int VarMax = FC_GetTableSize (ExprID);
  printf ("-----\nTABLE CONTAINS %d VARIABLES\n", VarMax);
  for (int VarID=0; VarID<VarMax; VarID++) {
    FC_GetVarName (ExprID, VarID, &varname);
    FC_GetVarString (ExprID, VarID, &str);
    printf ("%s = %s\n", varname, str);
  }
  FC_CalcString (ExprID, &str);      // вычисление выражения
  printf ("RESULT %s\n", str);       // печать результата
  if (FC_GetLastMessage (&str)==0);  // сообщ. об ошибке вычисления
    printf ("EXECUTION ERROR %s\n", str);
}

// построчное считывание файла, компиляция и вычисление выражений
void main () {
  FC_Start ();
  int ExprID = 0;
  char buf [2000];
  while (!feof(stdin)) {
    // считываем строку
    if (fgets (buf, 2000, stdin)==0) break;
    // пустые строки и комментарии игнорируем
    if (*buf=='\0' || *buf=='#' || *buf=='!')
      continue;
    // обрабатываем строку значений переменных последнего выражения
    if (*buf==';') {
      char* name = strtok (buf+1,"=");
      while (name) {
        char* value = strtok (0,";");
        if (name && value)
          FC_SetVarString (ExprID,FC_GetVarID(ExprID,name),value);
        name = strtok (0,"=");
      }
      calculate (ExprID);
      continue;
    }
    // обрабатываем строку с выражением
    compile (ExprID, buf);
    calculate (ExprID);
  }
  FC_Finish ();
  FC_String str;
  FC_GetMemoryReport (&str);
  printf ("\n============================\n%s\n", str);
}
```