ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
КАФЕДРА ИНФОРМАТИКИ И МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ

Отчет по курсу «Верификация ПО»

Выполнил:
студент 6 курса группы 22609 К. Н. Кирпиченок

Преподователь:
к.ф-м.н., доцент К. А. Кулаков

Петрозаводск – 2013

Оглавление

Объект тестирования

Mathex – библиотека для разбора/вычисления математических выражений, написанная на C++. (http://sscilib.sourceforge.net/, GNU LGPL)

Основные особенности:

- есть возможность добавления пользовательских переменных и функций
- поддерживает основные функции из библиотеки math.h (abs, acos, log и т.д.)
- обработка скобок и учет приоритета математических операций
- возможность использовать научную нотацию (напр. 1.7e+300)

Стратегия тестирования

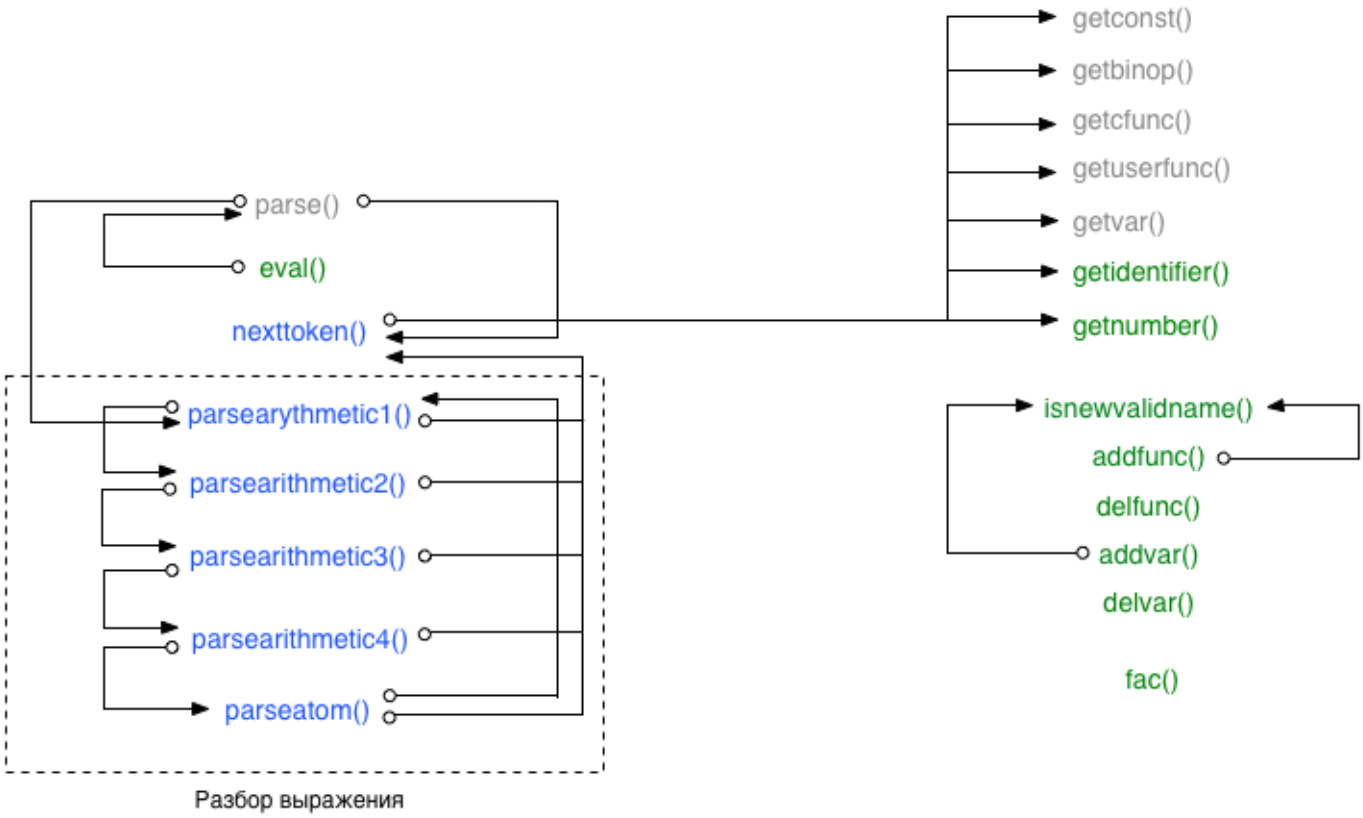

Рис. 1 Граф вызовов функций

- модульное тестирование
- интеграционное тестирование
- не тестируются

Для функций *getconst*, *getbinop*, *gecfunc*, *getuserfunc*, *getvar* писать блочные тесты не имеет смысла т.к. они просто возвращают индекс элемента в массиве. Также тесты не писались для простых математических функций (сложение, вычитание, перевод градусов в радианы и т.д.).

Функции *addfunc* и *addvar* проверяются при помощи блочных тестов, при этом для функции *isnewvalidname* используется заглушка (всегда возвращает истину).

Функцию *eval* также можно протестировать с использованием блочных тестов, установив переменную статуса разбора status в значение parsed и создав вектор bytecode вручную.

Функция *parse* отдельно не тестировалась т.к. в ней просто последовательно вызываются функции *nexttoken* и *parsearythmetic1*.

Описание классов

mathex – основной класс библиотеки, содержит несколько подклассов, используемых при разборе и вычислении выражения:

1. `CODETOKEN` – используется при вычислении выражения, содержит:
   - `state` – тип токена (`VALUE`, `FUNCTION`, `VARIABLE`, `BINOP`, `USERFUNC`)
   - `numargs` – количество аргументов функции, используется только для `FUNCTION/USERFUNC`
   - `value` – численное значение, используется для `VALUE`
   - `idx` – индекс переменной или функции в соответствующей таблице
2. `PARSERTOKEN` – используется при разборе выражения, структура аналогична классу `CODETOKEN`, но `state` может иметь дополнительные значения (`PLUS, MINUS, TIMES` и т.д.)
3. `FUNCREC` – используется для описания пользовательских функций
   - `name` – имя функции
   - `double (*f)(vector<double> const &x)` – указатель на функцию
   - `numargs` – количество аргументов
4. `VARREC` – используется для описания переменных
   - `name` – имя переменной
   - `double *var` – указатель на переменную
5. `CFUNCREC` – используется для описания функций из math.h
   - `name` – имя функции
   - `double (*f)(double x)` – указатель на функцию
6. `BINOPREC` – используется для описания бинарных математических операций
   - `name` – имя функции
   - `double (*f)(double, double)` – указатель на функцию

Глобальные переменные

- `vector<FUNCREC> functable` – вектор названий и указателей на пользовательские функции
- `vector<VARREC> vartable` – вектор названий и указателей на переменные
- `vector<CFUNCREC> cfunctable` – вектор названий и указателей на переменные из math.h
- `vector<BINOPREC> binoptable` – вектор названий и указателей на функции, соответствующие бинарным математическим операциям
- `vector<CODETOKEN> bytecode` – вектор, содержащий части разобранного выражения в виде `CODETOKEN'ов`
- `vector<double> evalstack` – стек, используемый при вычислении значения выражения (после вычисления на вершине стека находится значение текущего выражения)
- `string expr` – выражение, которое в данный момент разбирается, представленное в виде строки
- `enum {invalid, notparsed, parsed} status` – статус разбора текущего выражения
- `unsigned long pos` – индекс символа в выражении expr, на котором в текущий момент остановился разбор выражения
- `PARSERTOKEN curtok` – текущий разбираемый токен выражения

Описание тестируемых методов

1. `double fac(double x)` – расчет значения факториала для целого числа от 0 до 170
   - **Возвращаемое значение:** значение факториала, соответствующее аргументу. В случае ошибки создается исключение `smlib::mathex::error`.

2. `bool mathex::isnewvalidname(string const &name)` – проверка имени функции, константы или переменной на валидность.
   - Имя должно начинаться с буквы латинского алфавита или символа ‘_’, далее могут идти буквы латинского алфавита, цифры или символ ‘-‘.
   - **Возвращаемое значение:** истина если имя валидно, ложь в противном случае
3. `bool mathex::getnumber(double &x)` – данный метод определяет находится ли на позиции pos в выражении expr число или нет. Если на позиции pos находится число, то в переменную x кладется значение данного числа и указатель pos перемещается.
   - Число может быть целым или вещественным, а также записанным с использованием научной нотации (напр. 1.7e+304).
   - **Возвращаемое значение:** false – если на позиции pos выражения expr нет числа, true в противном случае
4. `bool mathex::getidentifier(string &name)` – данный метод определяет находится ли на позиции pos в выражении expr идентификатор или нет. Если на позиции pos находится идентификатор, то в переменную `name` помещается его имя и указатель pos перемещается.
   - Идентификатор может начинаться с буквы латинского алфавита либо символа ‘_’, далее могут идти символы латинского алфавита, цифры и символ ‘_’.
   - **Возвращаемое значение:** false – если на позиции pos выражения expr нет идентификатора, true в противном случае
5. `bool addfunc(string &name, double (*f)(vector<double> &), int NumArgs)` – добавляет функцию в таблицу функций functable
   - **Аргументы:**
     - name – имя функции
     - *f – указатель на функцию
     - NumArgs – количество аргументов функции, -1 соответсвтует неопределенному количеству аргументов
   - **Возвращаемое значение:** true – в случае успешного добавления, иначе false
6. `bool delfunc(string const &name)` – удаляет функцию из таблицы функций functable
   - **Возвращаемое значение:** true – в случае успешного удаления, иначе false

7. `bool addvar(string &name, double *var)` – добавляет переменную в таблицу переменных vartable
   - **Аргументы:**
     - name – имя переменной
     - *var – указатель на переменную
   - **Возвращаемое значение:** true – в случае успешного добавления, иначе false
8. `bool delvar(string const &name)` – удаляет переменную из таблицы переменных vartable
   - **Возвращаемое значение:** true – в случае успешного удаления, иначе false
9. `double nexttoken(double x)` – получение типа следующего токена в выражении expr
   - **Возвращаемое значение:** тип токена (`PARSERTOKEN`), в случае если токен не удалось определить возвращает `PARSERTOKEN::INVALID`
10. `void mathex::parse()` – выполняет разбор выражения expr и записывает результат в bytecode
11. `double mathex::eval()` – вычисляет значения выражения expr
12. `void mathex::parsearithmetic1(void)` – первый уровень разбора выражения, выполняет разбор операций сложения и вычитания
13. `void mathex::parsearithmetic2(void)` – выполняет разбор операций умножения, деления и деления по модулю
14. `void mathex::parsearithmetic3(void)` – выполняет разбор операции возведения в степень
15. `void mathex::parsearithmetic4(void)` – выполняет разбор унарного + и -

16. `void mathex::parseatom(void)` – выполняет разбор переменных, функций, чисел

Детальный план тестов

1. Блочные тесты

Метод `double fac(double x)`
Название тестового случая: `testNull`
Тип теста: краевой
Описание тестового случая: Тест проверяет правильность расчета значения факториала для нулевого значения
Ожидаемый результат: 1
Название тестового случая: `testNegative`
Тип теста: негативный
Описание тестового случая: Тест проверяет правильность работы метода fac для отрицательных значений аргумента
Ожидаемый результат: исключение `smlib::mathex::error`
Название тестового случая: `testOutOfRange`
Тип теста: негативный
Описание тестового случая: Тест проверяет правильность работы метода fac для значений аргумента больше 170
Ожидаемый результат: исключение `smlib::mathex::error`
Название тестового случая: `testCloseToMax`
Тип теста: краевой
Описание тестового случая: Тест проверяет правильность работы метода fac для максимально допустимого значения аргумента (170)
Ожидаемый результат: значение, соответствующее 170!
Название тестового случая: `testDouble`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность округления вещественного значения аргумента
Ожидаемый результат: пусть x – целое число от 0 до 170 значению аргумента x.5 должно соответсвовать значение (x + 1)! значению аргумента x.y ($0 \le y < 0.5$) должно соответсвовать значение x!
Название тестового случая: `testGeneral`
Тип теста: общий
Описание тестового случая: Тест проверяет правильность расчета значения факториала для целого числа x ($0 < x < 170$)
Ожидаемый результат: x!

Метод `bool mathex::isnewvalidname(string const &name)`

Название тестового случая: `testEmptyName`
Тип теста: краевой
Описание тестового случая: Тест проверяет правильность работы метода для значения аргумента в виде пустой строки
Ожидаемый результат: false

Название тестового случая: `testStartsWithNotLetter`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода для значений аргумента в виде строки, начинающейся с неразрешенного символа
Ожидаемый результат: false

Название тестового случая: `testSpecialCharacters`
Тип теста: негативный
Описание тестового случая: Тест проверяет правильность работы метода для значений аргумента в виде строки, в которой встречаются запрещенные символы
Ожидаемый результат: false

Название тестового случая: `testValidName`
Тип теста: общий
Описание тестового случая: Тест проверяет правильность работы метода для различных вариантов валидного значения аргумента
Ожидаемый результат:
значению аргумента “test” должен соответствовать результат true
значению аргумента “_test” должен соответствовать результат true
значению аргумента “test-test” должен соответствовать результат true
значению аргумента “test3” должен соответствовать результат true
значению аргумента “_test3-test” должен соответствовать результат true

Название тестового случая: `testBusyName`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода в случае если уже существует функция или переменная с таким именем
Начальные условия: необходимо добавить пользовательскую переменную “testvar” с использованием метода addvar и пользовательскую функцию “testfunc” с использованием метода addfunc
Ожидаемый результат:
значению аргумента “abs” должен соответствовать результат false
значению аргумента “pi” должен соответствовать результат false
значению аргумента “testvar” должен соответствовать результат false
значению аргумента “testfunc” должен соответствовать результат false

Метод `bool mathex::getnumber(double &x)`

Название тестового случая: `testEmptyExpression`
Тип теста: краевой
Описание тестового случая: Тест проверяет правильность работы метода если выражения expr является пустой строкой

Начальные условия: expr = “”, pos = 0
Ожидаемый результат: false, pos = 0

Название тестового случая: `testNotNumberExpression`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода в случае если на позиции pos в выражении expr нет числа
Начальные условия: pos = 0
Ожидаемый результат:
выражению expr = `"test1"` должен соответствовать результат false, pos = 0
выражению expr = `"+1"` должен соответствовать результат false, pos = 0
выражению expr = `"-1"` должен соответствовать результат false, pos = 0
выражению expr = `"."` должен соответствовать результат false, pos = 0

Название тестового случая: `testMultiNumberExpression`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода в случае если на позиции pos в выражении expr стоит целое число из нескольких цифр
Начальные условия: expr = “`12345`”, pos = 0
Ожидаемый результат: true, x = 12345, pos = 5

Название тестового случая: `testShortScientificForm`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода в случае если на позиции pos в выражении expr стоит число, записанное с использованием краткой научной нотации
Начальные условия: expr = “`.123`”, pos = 0
Ожидаемый результат: true, x = 0.123, pos = 4

Название тестового случая: `testNumberWithTextExpression`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода в случае если на позиции pos в выражении expr стоит число и после него или до него идут нечисловые токены
Ожидаемый результат:
выражению expr = `"1.1test"`, pos = 0 должен соответствовать результат true, pos = 3, x = 1.1
выражению expr = `"test1.1"`, pos = 4 должен соответствовать результат true, pos = 7, x = 1.1

Название тестового случая: `testDecimalNumber`
Тип теста: общий
Описание тестового случая: Тест проверяет правильность работы метода в случае если на позиции pos в выражении expr стоит вещественное число
Начальные условия: pos = 0
Ожидаемый результат:
выражению expr = `"12.305"` должен соответствовать результат true, pos = 6, x = 12.305
выражению expr = `"0.34"` должен соответствовать результат true, pos = 4, x = 0.34

Название тестового случая: `testScientificNotation`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода в случае если на позиции pos в выражении expr стоит число, записанное с использованием научной нотации
Начальные условия: pos = 0
Ожидаемый результат:

выражению expr = " 1.7e+3" должен соответствовать результат true, pos = 6, x = 1.7e+3
выражению expr = " 1.7e+301" должен соответствовать результат true, pos = 8, x = 1.7e+301
выражению expr = " 123.712e+3" должен соответствовать результат true, pos = 10, x = 123.712e+3
выражению expr = " 1.7e-3" должен соответствовать результат true, pos = 6, x = 1.7e-3
выражению expr = " 1.7E+3" должен соответствовать результат true, pos = 6, x = 1.7e+3
выражению expr = " 1.7e" должен соответствовать результат true, pos = 4, x = 1.7

Название тестового случая: testWrongScientificNotation
Тип теста: негативный
Описание тестового случая: Тест проверяет правильность работы метода в случае если на позиции pos в выражении expr стоит число, записанное с использованием научной нотации с ошибками
Начальные условия: pos = 0
Ожидаемый результат:
выражению expr = " 1.7e+A" должен соответствовать результат true, pos = 4, x = 1.7
выражению expr = " 1.7e+-1" должен соответствовать результат true, pos = 4, x = 1.7

Название тестового случая: testOutOfRangeNumbers
Тип теста: краевой
Описание тестового случая: Тест проверяет правильность работы метода в случае если на позиции pos в выражении expr стоит число, значение которого превышает MAX_DOUBLE
Начальные условия: pos = 0, expr = " 1.7e+320"
Ожидаемый результат: true, pos = 8, x = #INF

Метод `bool mathex::getidentifier(string &name)`

Название тестового случая: testEmptyExpression
Тип теста: краевой
Описание тестового случая: Тест проверяет правильность работы метода если выражения expr является пустой строкой
Начальные условия: expr = “”, pos = 0
Ожидаемый результат: false, pos = 0

Название тестового случая: testValidExpression
Тип теста: общий
Описание тестового случая: Тест проверяет правильность работы метода в случае если на позиции pos в выражении expr находится идентификатор
Начальные условия: pos = 0
Ожидаемый результат:
выражению expr = "a" должен соответствовать результат true, pos = 1, name = “a”
выражению expr = "test" должен соответствовать результат true, pos = 4, name = “test”
выражению expr = "_test" должен соответствовать результат true, pos = 5, name = “_test”
выражению expr = "te25st" должен соответствовать результат true, pos = 6, name = “te25st”
выражению expr = "test25" должен соответствовать результат true, pos = 6, name = “test25”
выражению expr = " test1_test_2" должен соответствовать результат true, pos = 12, name = “test1_test_2”

Название тестового случая: testNotValidExpression
Тип теста: негативный
Описание тестового случая: Тест проверяет правильность работы метода в случае если на позиции pos в выражении expr нет валидного идентификатора

Начальные условия: pos = 0

Ожидаемый результат:
выражению expr = `"1"` должен соответствовать результат false, pos = 0
выражению expr = `"1test"` должен соответствовать результат false, pos = 0
выражению expr = `"~test"` должен соответствовать результат false, pos = 0
выражению expr = `".test"` должен соответствовать результат false, pos = 0
выражению expr = `"_"` должен соответствовать результат false, pos = 0

Название тестового случая: `testExpressionWithOtherContent`

Тип теста: специальный

Описание тестового случая: Тест проверяет правильность работы метода в случае если на позиции pos в выражении expr стоит идентификатор, а до него или после него идут другие токены

Ожидаемый результат:
выражению expr = `"test.test2"`, pos = 0 должен соответствовать результат true, pos = 4, name = “test”
выражению expr = `"test test2"`, pos = 0 должен соответствовать результат true, pos = 4, name = “test”
выражению expr = `"_test_test3_1+test2"`, pos = 0 должен соответствовать результат true, pos = 13, name = “_test_test3_1”
выражению expr = `"_test_test3_1+test2"`, pos = 14 должен соответствовать результат true, pos = 19, name = “test2”

Метод `bool addfunc(string &name, double (*f)(vector<double> &), int NumArgs)`

Название тестового случая: `testAddingValidFunction`

Тип теста: общий

Описание тестового случая: Тест проверяет добавление новой корректной пользовательской функции test() без аргументов

Начальные условия: в качестве пользовательской функции используется тестовый метод p_test(), возвращающий 1

Ожидаемый результат:
функция успешно добавлена в functable
результат вычисления выражения expr = `"test()"` равен 1

Название тестового случая: `testOverwriteFunction`

Тип теста: специальный

Описание тестового случая: Тест проверяет перезапись существующей пользовательской функции max() другой пользовательской функцией.

Начальные условия: в качестве пользовательской функции используется тестовый метод p_test2(), возвращающий 2

Ожидаемый результат:
результат вычисления выражения expr = `"max()"` равен 2

Название тестового случая: `testInvalidArgsNum`

Тип теста: негативный

Описание тестового случая: Тест проверяет правильность работы метода при добавлении функции с некорректным количестов аргументов

Начальные условия: в качестве пользовательской функции используется тестовый метод p_test(), возвращающий 1

Ожидаемый результат: false

Название тестового случая: `testUnlimitedArgsNum`

Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода при добавлении функции test3() с неопределенным количестов аргументов
Начальные условия: в качестве пользовательской функции используется тестовый метод p_test3(), возвращающая сумму своих аргументов
Ожидаемый результат:
функция успешно добавлена в functable
результат вычисления выражения expr = `"test3(1,2,3,4,5)"` равен 15

Метод `bool delfunc(string const &name)`

Название тестового случая: `testDeletingNotExistingFunction`
Тип теста: негативный
Описание тестового случая: Тест проверяет удаление несуществующей функции
Ожидаемый результат: false

Название тестового случая: `testDeletingExistingFunction`
Тип теста: общий
Описание тестового случая: Тест проверяет удаление существующей функции
Ожидаемый результат: true

Метод `bool addvar(string &name, double *var)`

Название тестового случая: `testAddingValidVariable`
Тип теста: общий
Описание тестового случая: Тест проверяет добавление новой переменной
Начальные условия: в качестве переменной используется вещественная переменная x = 1.7
Ожидаемый результат:
переменная x успешно добавлена в vartable
результат вычисления выражения expr = `"x"` равен 1.7

Название тестового случая: `testOverwriteFunction`
Тип теста: специальный
Описание тестового случая: Тест проверяет добавление новой переменной с уже существующим именем в vartable
Начальные условия: в качестве существующей переменной используется вещественная переменная y = 1.7, в качестве новой переменной используется z = 0.3
Ожидаемый результат:
результат вычисления выражения expr = `"y"` равен 0.3

Название тестового случая: `testChangeVariableValue`
Тип теста: специальный
Описание тестового случая: Тест проверяет добавление новой переменной delta в vartable и последущее изменение значения данной переменной
Ожидаемый результат:
переменная delta успешно добавлена в vartable
при измененеии значения переменной delta результат вычисления выражения expr = `"delta"` должен соответсвовать значению переменной delta

Метод `bool delvar(string const &name)`

Название тестового случая: `testDeletingNotExistingVariable`
Тип теста: негативный
Описание тестового случая: Тест проверяет удаление несуществующей переменной
Ожидаемый результат: false

Название тестового случая: `testDeletingExistingVariable`
Тип теста: общий
Описание тестового случая: Тест проверяет удаление существующей переменной
Ожидаемый результат: true

Метод `double eval()`

Название тестового случая: `testInvalidStatus`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода, если статус разбора выражения установлен в значение invalid
Начальные условия: status = invalid
Ожидаемый результат: исключение `smlib::mathex::error`

Название тестового случая: `testVariables`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода при вычислении выражения, содержащего пользовательские переменные
Начальные условия: status = parsed, bytecode соответствует выражению “x/0.5” (x = 2)
Ожидаемый результат: 4

Название тестового случая: `testFunctions`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода при вычислении выражения, содержащего функции из math.h
Начальные условия: status = parsed, bytecode соответствует выражению ”cos(pi/2)”
Ожидаемый результат: 0

Название тестового случая: `testUserFunctionsWithNoArgs`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода при вычислении выражения, содержащего пользовательские функции без аргументов
Начальные условия: status = parsed, bytecode соответствует выражению “test()”, где test – пользовательская функция, возвращающая 1
Ожидаемый результат: 1

Название тестового случая: `testUserFunctionsWithMultipleArgs`
Тип теста: специальный
Описание тестового случая: Тест проверяет правильность работы метода при вычислении выражения, содержащего пользовательские функции с несколькими аргументами
Начальные условия: status = parsed, bytecode соответствует выражению “test2(1,2,3,4,5)”, где test2 – пользовательская функция, возвращающая сумму своих аргументов

Ожидаемый результат: 15

2. Интеграционные тесты

Метод `double nexttoken(double x)`

`Взаимодействие с getidentifier, getnumber, getbinop, getcfunc, getuserfunc, getvar, getconst`

Название тестового случая: `testEmptyExpression`
Описание тестового случая: Тест проверяет правильность работы метода если выражения expr является пустой строкой
Начальные условия: expr = “”
Ожидаемый результат: `PARSERTOKEN::END`

Название тестового случая: `testNumberExpression`
Описание тестового случая: Тест проверяет правильность работы метода если выражение expr содержит вещественное число
Начальные условия: expr = `"1.7e+3"`
Ожидаемый результат: `PARSERTOKEN::VALUE PARSERTOKEN::END` (*прим. далее в ожид. результате будет приводится последовательность типов токенов, соответсвующая последовательным вызовам метода nexttoken*)

Название тестового случая: `testInternalCFunction`
Описание тестового случая: Тест проверяет правильность работы метода если выражение expr содержит функцию из cfunctable
Начальные условия: expr = `"abs(1)"`
Ожидаемый результат: `FUNCTION OPAREN VALUE CPAREN END`

Название тестового случая: `testUserFunction`
Описание тестового случая: Тест проверяет правильность работы метода если выражение expr содержит функцию из functable
Начальные условия: expr = `"min(1,3)"`
Ожидаемый результат: `PARSERTOKEN::USERFUNC …`

Название тестового случая: `testComplexExpressionWithParenthesis`
Описание тестового случая: Тест проверяет правильность работы метода если выражение expr содержит сложное математической выражение с использованием скобок
Начальные условия: expr = `"(21.43-12)*2.1e+11"`
Ожидаемый результат: `OPAREN VALUE MINUS VALUE CPAREN TIMES VALUE END`

Название тестового случая: `testInvalidExpression`
Описание тестового случая: Тест проверяет правильность работы метода если выражение expr содержит несуществующий идентификатор
Начальные условия: expr = `"test_invalid(12)"`
Ожидаемый результат: `INVALID OPAREN VALUE CPAREN END`

Название тестового случая: `testConstants`
Описание тестового случая: Тест проверяет правильность работы метода если выражение expr содержит константы

Начальные условия: expr = `"pi/E"`
Ожидаемый результат: `VALUE DIVIDE VALUE END`

Название тестового случая: `testSpacesInExpression`
Описание тестового случая: Тест проверяет правильность работы метода если выражение expr содержит табуляцию и пробелы
Начальные условия: expr = `"    fac( 10 )   + 1"`
Ожидаемый результат: `FUNCTION OPAREN VALUE CPAREN PLUS VALUE END`

Название тестового случая: `testVariablesExpression`
Описание тестового случая: Тест проверяет правильность работы метода если выражение expr содержит пользовательские переменные
Начальные условия: с использованием метода addvar была добавлена пользовательская переменная x, expr = `"x/2"`
Ожидаемый результат: `VARIABLE DIVIDE VALUE END`

Группа тестов для тестирования цикла разбора выражения, представленного методами `parsearythmetic1, parsearythmetic2, parsearythmetic3, parsearythmetic4, parseatom + nexttoken`

Далее во всех тестах производится вызов метода `parsearythmetic1,` который в свою очередь вызывает `parsearythmetic2. parsearythmetic2` вызывает `parsearythmetic3 и т.д.`

Название тестового случая: `testDoubleValue`
Описание тестового случая: Тест проверяет правильность разбора выражения expr, содержащего вещественное число
Начальные условия: expr = “1.7”
Ожидаемый результат: вектор bytecode должен содержать единственный элемент `CODETOKEN(VALUE, 1.7)`

Название тестового случая: `testCFunction`
Описание тестового случая: Тест проверяет правильность разбора выражения expr, содержащего функцию из math.h
Начальные условия: expr = `"abs(10)"`
Ожидаемый результат: вектор bytecode должен иметь следующий вид
`[CODETOKEN(VALUE, 10), CODETOKEN(FUNCTION, индекс функции abs в cfunctable)]`

Название тестового случая: `testVariable`
Описание тестового случая: Тест проверяет правильность разбора выражения expr, содержащего переменную
Начальные условия: expr = `"x"`, где x – ранее добавленная с помощью addvar переменная
Ожидаемый результат: вектор bytecode должен иметь следующий вид
`[CODETOKEN(VARIABLE, индекс переменной x в vartable)]`

Название тестового случая: `testUserFunction`
Описание тестового случая: Тест проверяет правильность разбора выражения expr, содержащего пользовательскую функцию с несколькими аргументами
Начальные условия: expr = `"test(1,2)"`, где test – ранее добавленная с помощью addfunc функцию, возвращающая сумму своих аргументов
Ожидаемый результат: вектор bytecode должен иметь следующий вид
`[CODETOKEN(VALUE, 1), CODETOKEN(VALUE, 2), CODETOKEN(USERFUNC, индекс функции test в functable)]`

Название тестового случая: `testBinaryPlus`
Описание тестового случая: Тест проверяет правильность работы метода если выражение expr содержит операцию сложения двух чисел
Начальные условия: expr = `"1.9+10"`
Ожидаемый результат: вектор bytecode должен иметь следующий вид
`[CODETOKEN(VALUE, 1.9), CODETOKEN(VALUE, 10), CODETOKEN(BINOP, индекс операции сложения в binoptable)]`

Название тестового случая: `testDivision`
Описание тестового случая: Тест проверяет правильность работы метода если выражение expr содержит операцию деления двух чисел
Начальные условия: expr = `"3/2"`
Ожидаемый результат: вектор bytecode должен иметь следующий вид
`[CODETOKEN(VALUE, 3), CODETOKEN(VALUE, 2), CODETOKEN(BINOP, индекс операции деления в binoptable)]`

Название тестового случая: `testPower`
Описание тестового случая: Тест проверяет правильность работы метода если выражение expr содержит операцию возведения в степень
Начальные условия: expr = `"10^2"`
Ожидаемый результат: вектор bytecode должен иметь следующий вид
`[CODETOKEN(VALUE, 10), CODETOKEN(VALUE, 2), CODETOKEN(BINOP, индекс операции возведения в степень в binoptable)]`

Название тестового случая: `testUnaryMinus`
Описание тестового случая: Тест проверяет правильность работы метода если выражение expr содержит операцию унарного минуса
Начальные условия: expr = `"-1.9"`
Ожидаемый результат: вектор bytecode должен иметь следующий вид
`[CODETOKEN(VALUE, 1.9), CODETOKEN(FUNCTION, индекс операции унарного минуса в cfunctable)]`

Название тестового случая: `testPriority`
Описание тестового случая: Тест проверяет правильность учета приоритета операций при разборе выражения expr
Начальные условия: expr = `"25+10*2"`
Ожидаемый результат: вектор bytecode должен иметь следующий вид
`[CODETOKEN(VALUE, 25), CODETOKEN(VALUE, 10), CODETOKEN(VALUE, 2), CODETOKEN(BINOP, *), CODETOKEN(BINOP, +)]`
т.е. операция умножения должна идти раньше операции сложения

Название тестового случая: `testWrongParenthesis`
Описание тестового случая: Тест проверяет правильность работы метода в выражении expr нарушен баланс скобок
Начальные условия: expr = `"(10+2))"`, expr = `"min("`
Ожидаемый результат: исключение `smlib::mathex::error`

Название тестового случая: `testParenthesisPriority`
Описание тестового случая: Тест проверяет правильность учета приоритета операций при использовании скобок в выражении expr
Начальные условия: expr = `"(25+10)/2"`
Ожидаемый результат: вектор bytecode должен иметь следующий вид

```
[CODETOKEN(VALUE, 25), CODETOKEN(VALUE, 10), CODETOKEN(BINOP, +), CODETOKEN(VALUE, 2),
CODETOKEN(BINOP, /)]
```

3. Аттестационные тесты

1. Вычисление математического выражения, не содержащего функции и переменные, с использованием библиотеки mathex
2. Тестирование возможности добавления пользовательских функций и переменных
    - добавление пользовательской функции с использованием метода addfunc
    - добавление переменной с использованием addvar
    - вычисление выражения, содержащего пользовательскую функцию
    - вычисление выражения, содержащего добавленную переменную
3. Тестирование возможности использования функций из библиотеки math.h
    - вычисления выражения, содержащего функцию из math.h (например: log)
4. Тестирование учета приоритета операций и учета скобок
    - вычисление выражения, содержащего математические операции с различным приоритетом (напр. 10+2*3), операции *,/,%,^ должны иметь больший приоритет чем операции сложения и вычитания
    - вычисление выражения, использующего скобки для задания приоритета (напр. (100-20)/2)
5. Тестирование возможности использования научной нотации при записи чисел
    - вычисление выражения, содержащего числа записанные с использованием научной нотации (напр. 2+1.7E+30)

Примеры реализации тестов

Тесты реализовывались с использованием библиотеки Google Test (http://code.google.com/p/googletest/).

1. Пример реализации блочного теста `testNegative` для метода `double fac(double x)`

```
// инициализация библиотеки mathex
smlib::mathex mathex;

// реализация теста
// ASSERT_THROW позволяет выполнить проверку было ли выброшено исключение
TEST(testFac, testNegative)
{
    ASSERT_THROW(smlib::testing::fac(-1), smlib::mathex::error);
}

int main(int argc, char** argv)
{
     // инициализация библиотеки Google Test
     testing::InitGoogleTest(&argc, argv);

     // запуск тестов
     RUN_ALL_TESTS();
     system("pause"); // keep console window open until Return keystroke
}
```

2. Пример реализации интеграционного теста `testCFunction` для цикла разбора выражения

```
TEST(testParseCycle, testCFunction)
{
    mathex.expression("abs(10)");

    // необходимо вызвать nexttoken для получения типа первого токена выражения
    mathex.nexttoken();

    // начинаем с первого уровня разбора выражения
    mathex.parsearythmetic1();

    // проверка длины и содержимого вектора bytecode, содержащего разобранное
    // на CODETOKEN'ы выражения
    ASSERT_EQ(mathex.bytecode.size, 2);
    ASSERT_EQ(mathex.bytecode[0].state, smlib::mathex::CODETOKEN::VALUE);
    ASSERT_EQ(mathex.bytecode[0].value, 10);
    ASSERT_EQ(mathex.bytecode[1].state, smlib::mathex::CODETOKEN::FUNCTION);
    ASSERT_EQ(mathex.bytecode[1].idx, mathex.getcfunc('abs'));
}
```

Результаты тестирования

1. Блочные тесты

Метод	Количество тестов	Количество ошибок
`fac()`	6	1
`isnewvalidname()`	5	1
`getnumber()`	9	2
`getidentifier()`	5	1
`addfunc()`	4	2
`delfunc()`	2	1
`addvar()`	3	0
`delvar()`	2	1
`eval()`	5	0
Всего:	41	9

2. Интеграционные тесты

Метод	Количество тестов	Количество ошибок
`nexttoken()`	9	0
цикл разбора выражения	10	0
Всего:	19	0

3. Все аттестационные тесты были пройдены успешно.

Примеры найденных ошибок

1. Неправильное удаление пользовательских функций методом delfunc и переменных методом delvar

```
bool mathex::delfunc(string const &name)
{
   unsigned i;

   for(i=0; (i<functable.size()) && (functable[i].name != name);i++);
   if(i < functable.size())  {
      // vartable вместо functable + данный способ все равно не работал!
      for(unsigned j=0; j<vartable.size()-1; j++)
         functable[j] = functable[j+1];
      functable.pop_back(); // delete last
      return true;
   }
   else
      return false;
}
```

вариант исправления:

```
…
if(i < functable.size())  {
    functable.erase(functable.begin() + i, functable.begin() + i + 1);
    return true;
}
```

2. В ходе тестирования была обнаружена ошибка с возможным выходом за пределы массива в методе getnumber

```
// проверка не выходит ли индекс за пределы массива происходит после обращения по индексу
if((toupper(expr[i])=='E') && (i<expr.size()))
```

вариант исправления:

```
if((i<expr.size()) && (toupper(expr[i])=='E'))
```

3. Ошибка проверки первого символа имени в функции isnewvalidname

```
// последнее условие никогда не проверяется т.к. ‘_’ не является буквой
// в результате имена идентификаторов не могут начинаться с ‘_’
if(name.empty() || (!isalpha(name[0]) && (name[0] != '_')))
    return false;
```

вариант исправления:

```
if(name.empty() || ((name[0] != '_') && !isalpha(name[0])))
     return false;
```

4. Ошибка при добавлении функции с неправильным количеством аргументов

```
// возвращает истину, но функция не может содержать -2 аргумента!
mathex.addfunc("test", p_test, -2)
```

вариант исправления: добавить проверку на значение аргумента с количество параметров в функцию addfunc