Московский государственный институт радиотехники, электроники и автоматики (технический университет)

КУРСОВАЯ РАБОТА

на тему

" **СИНТАКСИЧЕСКИЙ АНАЛИЗ ВЫРАЖЕНИЯ.**

**ПАРСЕР МАТЕМАТИЧЕСКИХ ВЫРАЖЕНИЙ**"

по дисциплине

"Основы программирования (С++)"

Студент группы:

КМБ-1-11

Ананьев Роман

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель:

Петрусевич Денис Андреевич

Москва 2012

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Оглавление** | Стр. |
| 1) | Задание на курсовую……………………………………………………………... | | 3 |
| 2) | Описание возможностей работы программы………………………………….. | | 4 |
| 3) | Разбор и описание функций и описание методов, используемых в курсовой работе………………………………………………………………………………. | | 5 |
| 4) | Требования к входным данным, описание их формата и диапазона возможных значений…………………………………………………………….. | | 7 |
| 5) | Руководство по запуску программы с примерами и их результатами……….. | | 7 |
| 6) | Использованные материалы…………………………………………………….. | | 8 |
| 7) | Приложение 1. Листинг программы……………………………………………. | | 9 |
| 7.1) | Calc.cpp | | 9 |
| 7.2) | Ant1Freeze.h | | 10 |
| 7.3) | Ant1Freeze.cpp | | 13 |
| 8) | Приложение 2. Пример работы программы…………………………………… | | 24 |

**Задание на курсовую работу**

Приложение принимает от пользователя строку с арифметическим выражением, содержащим простейшие арифметические операции и скобки, а в ответ выводит результат вычислений.

\*Предусмотреть возможность ввода в выражение ограниченного списка функций (например, тригонометрических, возведения в степень, логарифмов и т.п.)

**Описание возможностей программы**

В своей курсовой работе я реализовал возможность получения от пользователя строку с арифметическим выражением, содержащим арифметические операции и скобки, а в ответ вывод результата вычислений.

В данной программе были реализованы следующие возможности:

* Работа с основными математическими операциями (+ - \* /)
* Работа со специфическими функциями (%, sqrt, ^)
* Работа с тригонометрическими функциями (sin, cos, arcsin, arccos, tg, ctg, arctg, arcctg)
* Работа с гиперболическими функциями (sh, ch, th, cth)
* Обработка функций (lg, ln, e^\_)
* Встроенные константы (g=9.81, pi=3.14, e=2.71)

**Разбор функций и описание методов, используемых в курсовой работе.**

Для ввода выражения в приложение я использую

«cin.getline(expr, 255)».

После ввода выражения программа начинает выполнять

«CALC.Compile(expr);» & «CALC.Evaluate();»

и выводит на экран ответ уже метод

«CALC.GetResult()».

Если же во время ввода была допущена ошибка, то «TError error» сделает своё дело и выведет на экран ошибку в формате «cerr << error.error << " на "<< error.pos << " месте " << endl << endl;» и продолжить работу, используя, «continue;»

Предметом особой гордости является обработчик ошибок

«TCALC::SendError»,

возвращающий различные ошибки, в зависимости от ситуации: если пропущена открывающая или закрывающая скобка, вы получите ошибку «Пропущена открывающая/закрывающая скобка» в зависимости от ситуации, если же в выражении попадается буква или слово, не соответствующие заданным функциям, вы получите ошибку «Что за хрень: '%s'», где на месте «%s» будет стоять строка «на error.pos месте», где «error.pos» - позиция с ошибкой (непонятного символа или открывающей / закрывающей скобки) или если выражение не введено, будет сообщено «Пустое выражение».

Для определения операций я написал

«TCALC::GetToken»

который переопределяет действия и опрелеляет, что именно выполняется в данном выражении. Например, если введен «+». То этот метод вернёт

«typToken = CALC\_PLUS;»,

что является действием сложения, определённым в методе

«TCALC::CalcTree»

и выполняет сложение двух ветвей древа

«(return CalcTree(tree->left)+CalcTree(tree->right);)».

И таким образом я переопределил все функции. Если же в выражении встречается буква или слово, то метод

«TCALC::GetToken»

определит и проверка

«while(IsLetter()) curToken[i++] = expr[pos++];»

вернёт что это за функция: будь то «Sin» или «ArCTg».

Для разбора выражения по скобкам и приоритету операций я реализовал метод «CALC.Compile»,

который собирает выражение до конца строки или “\0”.

Для построения самого древа в данной курсовой реализованы методы

«TCALCNode \*TCALC::Expr», «TCALCNode \*TCALC::Expr1»,

«TCALCNode \*TCALC::Expr2», «TCALCNode \*TCALC::Expr3»,

«TCALCNode \*TCALC::Expr4», «TCALCNode \*TCALC::Expr5».

Которые помимо того, что построят древо, дак еще и расставят в нём приоритеты как знаковые, так и если в выражении содержатся скобки.

В корне всего построенного древа лежит root, который определяется методом «TCALC::Evaluate»

и возвращает

«result = CalcTree(root); return result;»

что и является решением данного выражения.

Метод

«TCALC::CalcTree(TCALCNode \*tree)»,

написанный мною, переопределяет все действия таким образом, как они лежат в древе: «tree->left» и «tree->right», согласно общеприянятым правилам и законам.

# Требования к входным данным, описание их формата и диапазона возможных значений.

Пользователь вводит следующие данные:

Математическое выражение (поддерживаемое программой), которое может содержать как буквенное (и заглавные буквы и строчные), так и числовое значение типа double (-106<…<107)

# Допустимые функции и действия:

sin(), arcsin(), cos(), arcos(), tg(), arctg(), ctg(), arctg(), ch(), sh(), th(), cth(), ()%(), sqrt(), ()^(), ln(), lg(), exp(), +, -, \*, /, sqrt.

# Руководство по запуску программы с примерами и их результатами

1)Запускаем файл Kypca4.exe

2)Видим диалог с пользователем, предлагающий ввести выражение.

C:\Users\Ant1Freeze\Desktop\1.jpg

3)Вводим выражение.C:\Users\Ant1Freeze\Desktop\2.jpg

4)После ввода выражения нажимаем Enter.

5)На экран выводится ответ.

C:\Users\Ant1Freeze\Desktop\3.jpg

5.а) Пользователь может дальше проводить вычисления

5.б) Пользователь может выйти из программы, введя «bb», после чего нажав Enter. Программа завершит свою работу и закроется.

# Использованные материалы.

1. David Vandevoorde, Bjarne Stroustrup «C++ Solutions: Companion to the C++ Programming Language, 1998»
2. Ant1Freeze. «С++. Ветхий завет», 2011, Издательство: «Home»
3. Шилдт Г. «Самоучитель C++, 2002», Издательство: «БВХ-Петербург»
4. Стивен Дьюхерст, «C++. Священные знания, 2008», Издательство: Символ-Плюс
5. Либерти Д.  «Освой самостоятельно C++ за 21 день, 2006», Издательство: Вильямс
6. <http://www.programmersclub.ru/main/> - Учебник по C++ для начинающих
7. <http://www.codenet.ru/progr/cpp/sprd/> - Справочник по библиотечным функциям языка Си

**Приложение 1. Листинг программы.**

**Calc.cpp**

#include <cstring>

#include <iostream>

#include <windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <mmsystem.h>

#include <conio.h>

#include "Ant1Freeze.h"

#include <ios>

using namespace std;

int main(void)

{

setlocale (LC\_ALL, "Russian");

TCALC CALC;

char expr[255];

static double x[10];

HANDLE Ant1Freeze = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(Ant1Freeze, FOREGROUND\_INTENSITY | FOREGROUND\_GREEN);

cout << "Курсовая работа Ананьева Романа КМБ-1-11" << endl;

cout << "Введите выражение" << endl;

cout << "" << endl;

CALC.SetX((double \*)&x);

while(1)

{

//Ввод выражения в окно

cout << ">> ";

cin.getline(expr, 255);

// если "bb" то закрыться//////////////

if(!strcmp(expr, "bb")) break;//ВЫХОД//

///////////////////////////////////////

try

{

CALC.Compile(expr);

CALC.Evaluate();

cout << "Ответ: " << CALC.GetResult() << endl << endl; //Вывод ответа на экран

cout << "" <<endl;

}

catch(TError error)

{

cerr << error.error << " на "<< error.pos << " месте " << endl << endl; // Печать ошибки

continue;

}

}

return 0;

}

**Ant1Freeze.h**

#if !defined(\_\_SOLVER\_H)

#define \_\_SOLVER\_H

#include <ios>

#include <string.h>

using namespace std;

// Defines

#define MAX\_EXPR\_LEN 255

#define MAX\_TOKEN\_LEN 80

struct TCALCNode

{

double value;

TCALCNode \*left;

TCALCNode \*right;

TCALCNode(double \_value=0.0, TCALCNode \*\_left=NULL, TCALCNode \*\_right=NULL)

{

value = \_value;

left = \_left;

right = \_right;

}

};

struct TError

{

char \*error;

int pos;

TError() { };

TError(char \*\_error, int \_pos)

{

error=\_error;

pos=\_pos+1;

}

};

class TCALC

{

private:

TCALCNode \*root;

char \*expr;

char curToken[MAX\_TOKEN\_LEN];

enum {

CALC\_PLUS, CALC\_MINUS, CALC\_MULTIPLY, CALC\_DIVIDE, CALC\_PERCENT, ALC\_POWER,

CALC\_SIN, CALC\_COS, CALC\_TG, CALC\_CTG, CALC\_ARCSIN, CALC\_ARCCOS, ALC\_ARCTG, CALC\_ARCCTG, CALC\_SH, CALC\_CH, CALC\_TH, CALC\_CTH, CALC\_EXP, CALC\_LG, CALC\_LN, CALC\_SQRT, CALC\_X, CALC\_L\_BRACKET, CALC\_R\_BRACKET, CALC\_E, CALC\_PI, CALC\_NUMBER, CALC\_END, CALC\_G, ALC\_EXP1,CALC\_LEET

} typToken; //используемые действия, константы

int pos;

const double \*x;

double result;

private:

TCALCNode \*CreateNode(double \_value=0.0, TCALCNode \*\_left=NULL, TCALCNode \*\_right=NULL);

TCALCNode \*Expr(void);

TCALCNode \*Expr1(void);

TCALCNode \*Expr2(void);

TCALCNode \*Expr3(void);

TCALCNode \*Expr4(void);

TCALCNode \*Expr5(void);

bool GetToken(void);

bool IsDelim(void)

{

return (strchr("+-\*/%^()[]", expr[pos])!=NULL); //разрешенные символы

}

bool IsLetter(void)

{

return ((expr[pos]>='a' && expr[pos]<='z') ||

(expr[pos]>='A' && expr[pos]<='Z'));

}

bool IsDigit(void)

{

return (expr[pos]>='0' && expr[pos]<='9'); // разрешенные цифры

}

bool IsPoint(void)

{

return (expr[pos]=='.'); //и точка

}

double CalcTree(TCALCNode \*tree);

void DelTree(TCALCNode \*tree);

void SendError(int errNum);

public:

TCALC()

{

result = 0.0;

x = NULL;

root = NULL;

}

~TCALC()

{

DelTree(root);

root=NULL;

}

void SetX(const double \*\_x)

{

x=\_x;

}

bool Compile(char \*expr);

void Decompile()

{

DelTree(root);

root=NULL;

}

double Evaluate();

double Evaluate(double \*\_x)

{

SetX(\_x);

return Evaluate();

}

double Evaluate(double x, ...)

{

SetX(&x);

return Evaluate();

}

double GetResult(void)

{

return result;

}

};

#endif

**Ant1Freeze.cpp**

#include <math.h>

#include <ios>

#include "Ant1Freeze.h"

// OPerations

#define OP\_PLUS 0

#define OP\_MINUS 1

#define OP\_MULTIPLY 2

#define OP\_DIVIDE 3

#define OP\_PERCENT 4

#define OP\_POWER 5

#define OP\_UMINUS 6

#define OP\_SIN 7

#define OP\_COS 8

#define OP\_TG 9

#define OP\_CTG 10

#define OP\_ARCSIN 11

#define OP\_ARCCOS 12

#define OP\_ARCTG 13

#define OP\_ARCCTG 14

#define OP\_SH 15

#define OP\_CH 16

#define OP\_TH 17

#define OP\_CTH 18

#define OP\_EXP 19

#define OP\_LG 20

#define OP\_LN 21

#define OP\_SQRT 22

#define OP\_IN 23

#define OP\_X 24

// Определени математических констант

#define M\_PI 3.1415926535897932384626433832795

#define F\_G 9.81

#define M\_E 2.71828182845904523536

#define H\_LEET 1337

TCALCNode \*TCALC::CreateNode(double \_value, TCALCNode \*\_left, TCALCNode \*\_right)

{

TCALCNode \*pNode = new TCALCNode(\_value, \_left, \_right);

return pNode;

}

void TCALC::SendError(int errNum) //Обработчик ошибок

{

static char \*errs[10] =

{ NULL,

NULL,

"Внезапный конец выражения",

"Конец выражения ожидается",

"Пропущеннаи открывающая скобка",

"Пропущенна закрывающая скобка",

NULL

};

static char buffer[80];

int len = strlen(curToken);

if(\*curToken=='\0')

strcpy(curToken, "Пустое выражение");

switch(errNum)

{

case 0:

sprintf(buffer, "Что за хрень: '%s'", curToken);

errs[0] = buffer;

break;

case 1:

sprintf(buffer, "Что за хрень: '%s'", curToken);

errs[1] = buffer;

break;

case 6:

sprintf(buffer, "Что за хрень: '%s'", curToken);

errs[6] = buffer;

break;

}

TError error(errs[errNum], pos-len);

root = NULL;

throw error;

return;

}

bool TCALC::GetToken(void)

{

\*curToken = '\0';

while(expr[pos]==' ') pos++;

if(expr[pos]=='\0')

{

curToken[0] = '\0';

typToken = CALC\_END;

return true;

}

else

if(IsDelim())

{

curToken[0] = expr[pos++];

curToken[1] = '\0';

switch(\*curToken)

{

case '+':

typToken = CALC\_PLUS;

return true;

case '-':

typToken = CALC\_MINUS;

return true;

case '\*':

typToken = CALC\_MULTIPLY;

return true;

case '/':

typToken = CALC\_DIVIDE;

return true;

case '%':

typToken = CALC\_PERCENT;

return true;

case '^':

typToken = CALC\_POWER;

return true;

case '[':

case '(':

typToken = CALC\_L\_BRACKET;

return true;

case ']':

case ')':

typToken = CALC\_R\_BRACKET;

return true;

}

}

else if(IsLetter())

{

int i=0;

while(IsLetter()) curToken[i++] = expr[pos++];

curToken[i] = '\0';

int len = strlen(curToken);

for(i=0; i<len; i++)

if(curToken[i]>='A' && curToken[i]<='Z')

curToken[i] += 'a' - 'A';

if(!strcmp(curToken, "x"))

{

typToken = CALC\_X;

return true;

}

else if(!strcmp(curToken, "leet"))

{

typToken = CALC\_LEET;

return true;

}

else if(!strcmp(curToken, "e"))

{

typToken = CALC\_EXP1;

return true;

}

else if(!strcmp(curToken, "g"))

{

typToken = CALC\_G;

return true;

}

else if(!strcmp(curToken, "pi"))

{

typToken = CALC\_PI;

return true;

}

else if(!strcmp(curToken, "sin"))

{

typToken = CALC\_SIN;

return true;

}

else if(!strcmp(curToken, "cos"))

{

typToken = CALC\_COS;

return true;

}

else if(!strcmp(curToken, "tg"))

{

typToken = CALC\_TG;

return true;

}

else if(!strcmp(curToken, "ctg"))

{

typToken = CALC\_CTG;

return true;

}

else if(!strcmp(curToken, "arcsin"))

{

typToken = CALC\_ARCSIN;

return true;

}

else if(!strcmp(curToken, "arccos"))

{

typToken = CALC\_ARCCOS;

return true;

}

else if(!strcmp(curToken, "arctg"))

{

typToken = CALC\_ARCTG;

return true;

}

else if(!strcmp(curToken, "arcctg"))

{

typToken = CALC\_ARCCTG;

return true;

}

else if(!strcmp(curToken, "sh"))

{

typToken = CALC\_SH;

return true;

}

else if(!strcmp(curToken, "ch"))

{

typToken = CALC\_CH;

return true;

}

else if(!strcmp(curToken, "th"))

{

typToken = CALC\_TH;

return true;

}

else if(!strcmp(curToken, "cth"))

{

typToken = CALC\_CTH;

return true;

}

else if(!strcmp(curToken, "exp"))

{

typToken = CALC\_EXP;

return true;

}

else if(!strcmp(curToken, "lg"))

{

typToken = CALC\_LG;

return true;

}

else if(!strcmp(curToken, "ln"))

{

typToken = CALC\_LN;

return true;

}

else if(!strcmp(curToken, "sqrt"))

{

typToken = CALC\_SQRT;

return true;

}

else SendError(0);

}

else if(IsDigit() || IsPoint())

{

int i=0;

while(IsDigit()) curToken[i++] = expr[pos++];

if(IsPoint())

{

curToken[i++] = expr[pos++];

while(IsDigit()) curToken[i++] = expr[pos++];

}

curToken[i] = '\0';

typToken = CALC\_NUMBER;

return true;

}

else

{

curToken[0] = expr[pos++];

curToken[1] = '\0';

SendError(1);

}

return false;

}

bool TCALC::Compile(char \*\_expr)

{

pos = 0;

expr = \_expr;

\*curToken = '\0';

if(root!=NULL)

{

DelTree(root);

root = NULL;

}

GetToken();

if(typToken==CALC\_END) SendError(2);

root = Expr();

if(typToken!=CALC\_END) SendError(3);

return true;

}

TCALCNode \*TCALC::Expr(void)

{

TCALCNode \*temp = Expr1();

while(1)

{

if(typToken == CALC\_PLUS)

{

GetToken();

temp = CreateNode(OP\_PLUS, temp, Expr1());

}

else if(typToken==CALC\_MINUS)

{

GetToken();

temp = CreateNode(OP\_MINUS, temp, Expr1());

}

else break;

}

return temp;

}

TCALCNode \*TCALC::Expr1(void)

{

TCALCNode \*temp = Expr2();

while(1)

{

if(typToken==CALC\_MULTIPLY)

{

GetToken();

temp = CreateNode(OP\_MULTIPLY, temp, Expr2());

}

else if(typToken==CALC\_DIVIDE)

{

GetToken();

temp = CreateNode(OP\_DIVIDE, temp, Expr2());

}

else if(typToken==CALC\_PERCENT)

{

GetToken();

temp = CreateNode(OP\_PERCENT, temp, Expr2());

}

else break;

}

return temp;

}

TCALCNode \*TCALC::Expr2(void)

{

TCALCNode \*temp = Expr3();

while(1)

{

if(typToken==CALC\_POWER)

{

GetToken();

temp = CreateNode(OP\_POWER, temp, Expr2());

}

else break;

}

return temp;

}

TCALCNode \*TCALC::Expr3(void)

{

TCALCNode \*temp;

if(typToken==CALC\_PLUS)

{

GetToken();

temp = Expr4();

}

else if(typToken==CALC\_MINUS)

{

GetToken();

temp = CreateNode(OP\_UMINUS, Expr4());

}

else

temp = Expr4();

return temp;

}

TCALCNode \*TCALC::Expr4(void)

{

TCALCNode \*temp;

if(typToken >= CALC\_SIN && typToken<=CALC\_X)

{

temp = CreateNode(OP\_SIN-CALC\_SIN+typToken);

GetToken();

if(typToken!=CALC\_L\_BRACKET)

SendError(4);

GetToken();

temp->left = Expr();

if(typToken!=CALC\_R\_BRACKET)

SendError(5);

GetToken();

}

else

temp = Expr5();

return temp;

}

TCALCNode \*TCALC::Expr5(void)

{

TCALCNode \*temp;

switch(typToken)

{

case CALC\_NUMBER:

temp = CreateNode((double)atof(curToken));

GetToken();

break;

case CALC\_PI:

temp = CreateNode((double)M\_PI);

GetToken();

break;

case CALC\_G:

temp =CreateNode((double)F\_G);

GetToken();

break;

case CALC\_L\_BRACKET:

GetToken();

temp = Expr();

if(typToken!=CALC\_R\_BRACKET) SendError(5);

GetToken();

break;

case CALC\_EXP1:

temp = CreateNode((double) M\_E);

GetToken();

break;

case CALC\_LEET:

temp = CreateNode((double) H\_LEET);

GetToken();

break;

default:

SendError(6);

}

return temp;

}

double TCALC::Evaluate(void)

{

result = CalcTree(root);

return result;

}

double TCALC::CalcTree(TCALCNode \*tree) // Обработчик действий

{

static double temp;

if(tree->left==NULL && tree->right==NULL)

return tree->value;

else

switch((int)tree->value)

{

case OP\_PLUS:

return CalcTree(tree->left)+CalcTree(tree->right);

// "(tree->left)+(tree->right)"

case OP\_MINUS:

return CalcTree(tree->left)-CalcTree(tree->right);

// "(tree->left)-(tree->right)"

case OP\_MULTIPLY:

return CalcTree(tree->left)\*CalcTree(tree->right);

// "(tree->left)\*(tree->right)"

case OP\_DIVIDE:

return CalcTree(tree->left)/CalcTree(tree->right);

// "(tree->left)/(tree->right)"

case OP\_PERCENT:

return (int)CalcTree(tree->left)%(int)CalcTree(tree->right);

// "(tree->left)%(tree->right)"

case OP\_POWER:

return (double)pow(CalcTree(tree->left),CalcTree(tree->right));

// "(tree->left)^(tree->right)"

case OP\_UMINUS:

return -CalcTree(tree->left);

// "-(tree->left)"

case OP\_SIN:

return sin(CalcTree(tree->left));

// "sin(tree->left"

case OP\_COS:

return cos(CalcTree(tree->left));

// "cos(tree->left)"

case OP\_TG:

return tan(CalcTree(tree->left));

// "tan(tree->left)"

case OP\_CTG:

return 1.0/tan(CalcTree(tree->left));

// "ctg(tree->left)"

case OP\_ARCSIN:

return asin(CalcTree(tree->left));

// "arcsin(tree->left)"

case OP\_ARCCOS:

return acos(CalcTree(tree->left));

// "arccos(tree->left)"

case OP\_ARCTG:

return atan(CalcTree(tree->left));

// "atctg(tree->left)"

case OP\_ARCCTG:

return M\_PI/2.0-atan(CalcTree(tree->left));

// "arcctg(tree->left)"

case OP\_SH:

temp = CalcTree(tree->left);

return (exp(temp)-exp(-temp))/2.0;

// "sh(tree->left)"

case OP\_CH:

temp = CalcTree(tree->left);

return (exp(temp)+exp(-temp))/2.0;

// "ch(tree->left)"

case OP\_TH:

temp = CalcTree(tree->left);

return (exp(temp)-exp(-temp))/(exp(temp)+exp(-temp));

// "th(tree->left)"

case OP\_CTH:

temp = CalcTree(tree->left);

return (exp(temp)+exp(-temp))/(exp(temp)-exp(-temp));

// "cth(tree->left)"

case OP\_EXP:

return exp(CalcTree(tree->left));

// "e^(tree->left)"

case OP\_LG:

return log10(CalcTree(tree->left));

// "log10(tree->left)"

case OP\_LN:

return log(CalcTree(tree->left));

// "loge(tree->left)"

case OP\_SQRT:

return sqrt(CalcTree(tree->left));

// "(tree->left)^(1/2)"

case OP\_X:

return x[int(CalcTree(tree->left))];

case OP\_IN:

return 1;

}

return 0;

}

void TCALC::DelTree(TCALCNode \*tree)

{

if(tree==NULL) return;

DelTree(tree->left);

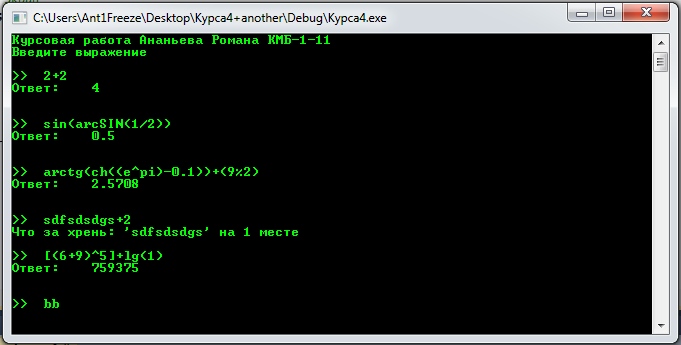
DelTree(tree->right);

delete tree;

return;

}

**Приложение 2. Пример работы программы.**

****