#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <string>

#include <sstream>

#include <cassert>

#include <cmath>

using namespace std;

// Тип числа

enum Type { PLUS, MIN, MUL, DIV, SQU, SIN, COS, TG, CTG, VAL };

// Структура дерево

struct Tree

{

Type type;

Tree\* left, \* right;

string data;

};

// Поиск минимума и максимума на отрезке

void find\_min\_max(Tree\* tree, string s);

// Вычисление значения

double calculate(Tree\* tree, double val);

// Создание дерева

void make\_tree(string str, Tree\* node);

// Удаление дерева

void clear\_tree(Tree\* tree);

// Основная программа

int main()

{

string s("None");

Tree\* tree = nullptr;

while (true)

{

cout << "Choose a command: " << endl;

cout << "0 - exit" << endl;

cout << "1 - input function" << endl;

cout << "2 - find min and max in the segment" << endl;

cout << "3 - print the current function" << endl;

cout << "\nYour answer: ";

char ans;

cin >> ans;

cout << endl;

cin.ignore(2, '\n');

switch (ans)

{

case '0':

return 0;

break;

case '1':

clear\_tree(tree);

cout << "Input function y = ";

getline(cin, s);

tree = new Tree;

make\_tree(s, tree);

break;

case '2':

if (tree != nullptr)

find\_min\_max(tree, s);

else

cout << "Please enter the function" << endl;

break;

case '3':

cout << "Current function is y = " << s << endl;

break;

default:

cout << "Command " << ans <<" is unknown" << endl;

}

cout << endl;

system("pause");

system("cls");

}

return 0;

}

void find\_min\_max(Tree\* tree, string s)

{

double a, b, n;

cout << "Input the ends of segment [a; b]: ";

cin >> a >> b;

cout << "Input accuracy (n < 1): ";

cin >> n;

double h = (b - a) \* 1.0 / n, min, max, min\_x = a, max\_x = a;

min = max = calculate(tree, a);

for (double x = a; x <= b; x += n)

{

double c = calculate(tree, x);

if (c > max) { max = c; max\_x = x; }

if (c < min) { min = c; min\_x = x; }

}

cout << "\nFunction y = " << s << endl;

cout << "In the segment [" << a << "; " << b << "]: " << endl << "min = y(" << min\_x << ") = "<< min << "; max = y(" << max\_x << ") = "<< max << endl;

}

double calculate(Tree\* tree, double val)

{

double res = 0;

switch (tree->type)

{

case PLUS:

res = calculate(tree->left, val) + calculate(tree->right, val);

break;

case MIN:

res = calculate(tree->left, val) - calculate(tree->right, val);

break;

case MUL:

res = calculate(tree->left, val) \* calculate(tree->right, val);

break;

case DIV:

if (calculate(tree->right, val) != 0)

res = calculate(tree->left, val) / calculate(tree->right, val);

else

{

cerr << "Divison by 0" << endl;

exit(1);

}

break;

case SQU:

res = pow(calculate(tree->left, val), calculate(tree->right, val));

break;

case SIN:

res = sin (calculate(tree->left, val));

break;

case COS:

res = cos(calculate(tree->left, val));

break;

case TG:

res = tan(calculate(tree->left, val));

break;

case CTG:

res = 1/(tan(calculate(tree->left, val)));

break;

case VAL:

if (tree->data == "x")

res = val;

else if (tree->data == "pi")

res = M\_PI;

else if (tree->data == "e")

res = M\_E;

else

res = atoi(tree->data.c\_str());

break;

default:

//assert(0);

return 0;

}

return res;

}

void make\_tree(string s, Tree\* tree)

{

unsigned i, c;

cout << s << std::endl;

for (unsigned t = PLUS; t < VAL; ++t)

for (i = 0, c = 0; i < s.size(); ++i)

{

if ('(' == s[i])

++c;

else if (')' == s[i])

--c;

else if ((t == PLUS ? '+' : t == MIN ? '-': t == DIV ? '/': '\*') == s[i] && !c)

{

tree->left = new Tree;

tree->right = new Tree;

tree->type = static\_cast<Type>(t);

make\_tree(s.substr(0, i), tree->left);

make\_tree(s.substr(i + 1, s.size() - i), tree->right);

return;

}

else if (t == SQU && s[i] == '^' && !c)

{

unsigned j, c1;

tree->left = new Tree;

tree->right = new Tree;

tree->type = static\_cast<Type>(t);

make\_tree(s.substr(0, i), tree->left);

for (j = i + 1, c1 = 0; j < s.size() && (c1 || isdigit(s[j]) || s[j] == '('); ++j)

{

if ('(' == s[j])

++c1;

else if (')' == s[j])

--c1;

}

make\_tree(s.substr(i + 1, j - i - 1), tree->right);

return;

}

else if (s.size() - i >= 3)

{

if (((t == SIN ? "sin" : t == COS ? "cos" : "ctg") == s.substr(i, 3) || (s.substr(i, 2) == "tg" && t == TG)) && !c)

{

tree->left = new Tree;

tree->right = nullptr;

tree->type = static\_cast<Type>(t);

if (t == SIN || t == COS || t == CTG)

{

unsigned j, c1;

for (j = i + 3, c1 = 0; j < s.size() && (c1 || s[j] == '('); ++j)

{

if ('(' == s[j])

++c1;

else if (')' == s[j])

--c1;

}

make\_tree(s.substr(i + 3, j - i - 1), tree->left);

}

else

{

unsigned j, c1;

for (j = i + 2, c1 = 1; j < s.size() && c1; ++j)

{

if ('(' == s[j])

++c1;

else if (')' == s[j])

--c1;

}

make\_tree(s.substr(i + 2, j - i - 1), tree->left);

}

return;

}

}

}

if (s[0] == '(' && s[s.size() - 1] == ')')

{

make\_tree(s.substr(1, s.size() - 2), tree);

return;

}

tree->type = VAL;

// Копирование данных

stringstream istr(stringstream::in | stringstream::out);

istr << s;

istr >> tree->data;

tree->left = nullptr;

tree->right = nullptr;

}

void clear\_tree(Tree\* tree)

{

if (!tree) return;

if (tree->left) clear\_tree(tree->left);

if (tree->right) clear\_tree(tree->right);

delete tree;

}