|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет  имени Н. Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н. Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | «Информатика, искусственный интеллект и системы управления» (ИУ) |

|  |  |
| --- | --- |
| КАФЕДРА | «Информационная безопасность» (ИУ8) |

Домашняя работа № 3

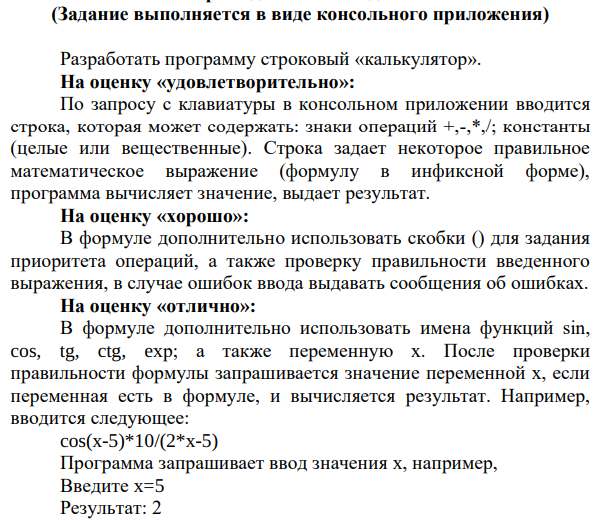
ПО КУРСУ

«Алгоритмические языки»

Программа «Калькулятор»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ИУ8-12 |  |  |  | В. К. Терехов |
|  | (Группа) |  |  |  | (И. О. Фамилия) |
|  |  |  |  |  |  |
| Преподаватель: |  |  |  |  | В. В. Соборова |
|  |  |  |  |  | (И.О. Фамилия) |

**Вариант 29**

**Задание:** 

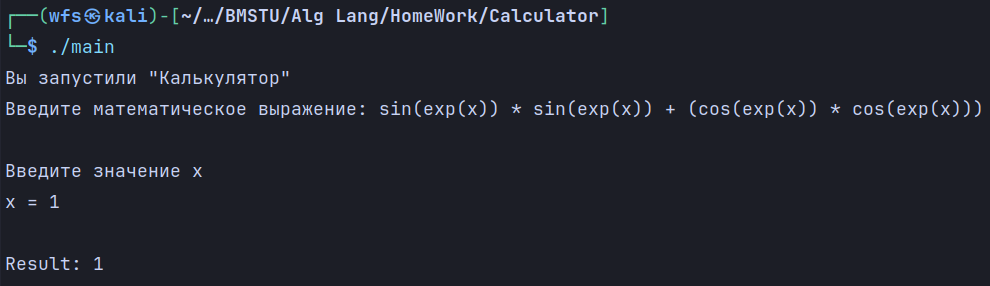
**Код:**

#include <iostream>  
#include <stack>  
#include <list>  
#include <cmath>  
#include <string>  
#include <cctype>  
#include <cstdlib>  
  
*using namespace* std;  
  
// Структура для представления лексемы  
*struct* Leksema {  
 *char* type; // '0' - число, 'f' - функция, операторы, скобки  
 *double* val; // Значение числа  
 string func; // Название функции  
};  
  
// Получаем приоритет для оператора или функции  
*int* getRang(Leksema &item) {  
 *if* (item.type == 'f') *return* 3;  
 *if* (item.type == '\*' || item.type == '/') *return* 2;  
 *if* (item.type == '+' || item.type == '-') *return* 1;  
 *return* 0;  
}  
  
*bool* isValidFunction(*const* string& func) {  
 *return* (func == "sin" || func == "cos" || func == "tan" || func == "ctg" || func == "exp");  
}  
  
// Функция для вычисления значения тригонометрической функции  
*double* calcFunction(*const* string &func, *double* x) {  
 *if* (func == "sin") *return* sin(x);  
 *else if* (func == "cos") *return* cos(x);  
 *else if* (func == "tan") *return* tan(x);  
 *else if* (func == "ctg") *return* 1 / tan(x);  
 *else if* (func == "exp") *return* exp(x);  
 *else* {  
 cerr << "ERROR!\nНеизвестная функция: " << func << endl;  
 exit(1); // из-за того, что в main не получиться передать то, что не надо выполнять прогу дальше  
 }  
}  
  
// Функция для преобразования инфиксного выражения в постфиксное  
*void* infixToPostfix(*const* string &input, list<Leksema> &myqueue, *double* x\_value, *bool* flagx) {  
 string curT; // переменная для буквенных  
 *bool* fl = *false*; // флаг для унарного минуса  
 stack<Leksema> mystack;  
  
 *for* (size\_t i = 0; i < input.size(); ++i) {  
 *char* Ch = input[i];  
  
 // Пропуск пробелов  
 *if* (isspace(Ch)) *continue*;  
  
 // Чтение числа  
 *if* (isdigit(Ch) || Ch == '.') {  
 curT += Ch;  
 *if* (i == input.size() - 1 || !isdigit(input[i + 1]) && input[i + 1] != '.') {  
 // добавляем в стек переменную структуры Leksema ЧИСЛО  
 Leksema item;  
 item.type = 0;  
 item.val = stod(curT);  
 myqueue.push\_back(item);  
 curT.clear();  
 }  
 }// Обработка переменной x  
 *else if* (Ch == 'x' && input[i + 1] != 'p' && input[i - 1] != 'e' && flagx) {  
 // если есть ч  
 Leksema item;  
 item.type = 0;  
 item.val = x\_value;  
 myqueue.push\_back(item);  
 }// Обработка операций  
 *else if* (Ch == '+' || Ch == '-' || Ch == '\*' || Ch == '/') {  
 *if* (Ch == '-' && (fl || i == 0 || input[i-1] == '('  
 || input[i-1] == '+' || input[i-1] == '-' || input[i-1] == '\*' || input[i-1] == '/')) {  
 // Унарный минус  
 curT += Ch;  
 fl = *false*;  
 *continue*;  
 }  
  
 Leksema operIt;  
 operIt.type = Ch;  
  
 *while* (!mystack.empty() && getRang(mystack.top()) >= getRang(operIt)) {  
 myqueue.push\_back(mystack.top());  
 mystack.pop();  
 }  
  
 mystack.push(operIt);  
 } // Обработка скобок  
 *else if* (Ch == '(') {  
 Leksema item;  
 item.type = Ch;  
 mystack.push(item);  
 }  
 *else if* (Ch == ')') {  
 *while* (!mystack.empty() && mystack.top().type != '(') {  
 myqueue.push\_back(mystack.top());  
 mystack.pop();  
 }  
 mystack.pop();  
 } // sin, cos, tan, ctg, exp  
 *else if* (isalpha(Ch)) {  
 curT += Ch;  
 // Проверка на завершение названия функции  
 *if* (isValidFunction(curT)) {  
 // Если функция завершена, добавляем её в стек  
 Leksema item;  
 item.type = 'f';  
 item.func = curT;  
 mystack.push(item);  
 curT.clear();  
 } *else if* (curT.length() > 3) { // Если слишком длинное имя и не валидно  
 cerr << "ERROR!\nНеизвестная функция: " << curT << endl;  
 exit(1);  
 }  
 }  
 fl = *false*;  
 }  
  
 // Переносим остатки из стека в очередь  
 *while* (!mystack.empty()) {  
 myqueue.push\_back(mystack.top());  
 mystack.pop();  
 }  
}  
  
// Вычисление значения выражения в постфиксной форме  
*double* calculateResult(list<Leksema> &myqueue) {  
 stack<Leksema> mystack;  
  
 *while* (!myqueue.empty()) {  
 Leksema item = myqueue.front();  
 myqueue.pop\_front();  
  
 *if* (item.type == 0) {  
 mystack.push(item);  
 }  
 *else if* (item.type == 'f') {  
 *if* (mystack.empty()) {  
 cerr << "ERROR!\nВыход за нижнюю границу стека" << endl;  
 *return* 0;  
 }  
 // подсчет функции  
 Leksema operand = mystack.top();  
 mystack.pop();  
 operand.val = calcFunction(item.func, operand.val);  
 mystack.push(operand);  
 }  
 *else* { // Операция  
 *if* (mystack.size() < 2) {  
 cerr << "ERROR!\nНедостаточно операндов" << endl;  
 *return* 0;  
 }  
  
 Leksema operand2 = mystack.top();  
 mystack.pop();  
 Leksema operand1 = mystack.top();  
 mystack.pop();  
  
 *double* result = 0;  
 *switch* (item.type) {  
 *case* '+': result = operand1.val + operand2.val; *break*;  
 *case* '-': result = operand1.val - operand2.val; *break*;  
 *case* '\*': result = operand1.val \* operand2.val; *break*;  
 *case* '/':  
 *if* (operand2.val == 0) {  
 cerr << "ERROR!\nДеление на ноль только в матане";  
 exit(1);  
 }  
 result = operand1.val / operand2.val;  
 *break*;  
 *default*:  
 cerr << "ERROR!\nНеправильный оператор" << endl;  
 *return* 0;  
 }  
  
 operand1.val = result; // Результат операции  
 mystack.push(operand1);  
 }  
 }  
  
 *if* (mystack.size() == 1) {  
 *return* mystack.top().val;  
 } *else* {  
 cerr << "ERROR!\nВыражение введено неправильно" << endl;  
 *return* 0;  
 }  
}  
  
*bool* xInEXpression(string str) {  
 string fStr = "";  
 *for* (*int* i = 0; i < str.length(); i++) {  
 *if* (str[i] == 'x' && str[i - 1] != 'e' && str[i + 1] != 'x') *return true*;  
 }  
 *return false*;  
}  
  
*int* main() {  
  
 string input;  
 *double* x\_value = 0;  
 *bool* x = *false*;  
  
 cout << "Вы запустили \"Калькулятор\"\nВведите математическое выражение: ";  
 getline(cin, input);  
 cout << endl;  
  
 // Проверка наличия x  
 *if* (input.find('x') != string::npos && xInEXpression(input)) {  
 x = *true*;  
 cout << "Введите значение x\nx = ";  
 cin >> x\_value;  
 cout << endl;  
 }  
  
 list<Leksema> myqueue;  
 infixToPostfix(input, myqueue, x\_value, xInEXpression(input));  
  
  
 cout << "Постфиксная запись: ";  
 *for* (*auto* &item : myqueue) {  
 *if* (item.type == 0) cout << item.val << ' ';  
 *else if* (item.type == 'f') cout << item.func << ' ';  
 *else* cout << item.type << ' ';  
 }  
 cout << endl;  
  
  
 *double* result = calculateResult(myqueue);  
  
 cout << "Результат: " << result << endl;  
  
 *return* 0;  
}

cout << "Постфиксная запись: ";  
 for (auto &item : myqueue) {  
 if (item.type == 0) cout << item.val << ' ';  
 else if (item.type == 'f') cout << item.func << ' ';  
 else cout << item.type << ' ';  
 }  
 cout << endl;  
 \*/  
  
 *double* result = calculateResult(myqueue);  
  
 cout << "Результат: " << result << endl;  
  
 *return* 0;  
}

**Консоль и контрольный расчет:**

****

**Вывод:**

В ходе выполнения домашней работы я разработал калькулятор.