**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: Рекурсия**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Карамышев А.О. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

**Цель работы:** ознакомиться с основными понятиями и приёмами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций на языке программирования C++.

**Постановка задачи:** предлагается написать программу, которая по *константному\_выражению* вычисляет его значение либо сообщает о переполнении (превышении заданного значения) в процессе вычислений. *Константное\_выражение* имеет вид:

*константное\_выражение::=ряд\_цифр|константное\_выражение знак\_операции константное\_выражение*

*знак\_операции::=+ | − | \**

*ряд\_цифр::=цифра | цифра ряд\_цифр*

**Теоретические сведения:** в C++ функции могут вызывать сами себя. Функция является рекурсивной, если оператор в теле функции вызывает функцию, содержащую данный оператор. Иногда называемая круговым определением, рекурсия является процессом определения чего-либо с использованием самой себя.

Когда функция вызывает сама себя, в стеке выделяется место для новых локальных переменных и параметров. Код функции работает с данными переменными. Рекурсивный вызов не создает новую копию функции. Новыми являются только аргументы. Поскольку каждая рекурсивно вызванная функция завершает работу, то старые локальные переменные и параметры удаляются из стека и выполнение продолжается с точки, в которой было обращение внутри этой же функции. Рекурсивные функции вкладываются одна в другую как элементы подзорной трубы.

Рекурсивные версии большинства подпрограмм могут выполняться немного медленнее, чем их итеративные эквиваленты, поскольку к необходимым действиям добавляются вызовы функций. Но в большинстве случаев это не имеет значения. Много рекурсивных вызовов в функции может привести к переполнению стека. Поскольку местом для хранения параметров и локальных переменных функции является стек и каждый новый вызов создает новую копию переменных, пространство стека может исчерпаться. Если это произойдет, то возникнет ошибка - переполнение стека.

Основным преимуществом применения рекурсивных функций является использование их для более простого создания версии некоторых алгоритмов по сравнению с итеративными эквивалентами. Например, сортирующий алгоритм Quicksort достаточно трудно реализовать итеративным способом. Некоторые проблемы, особенно связанные с искусственным интеллектом, также используют рекурсивные алгоритмы.

**Спецификация функций:**

bool num (istream& in, int& res) – функция считывающая число с потока и записывающая ее значение в переменную res, возвращает в зависимости от входящего числа значение true или false (в зависимости от того, вызывает ли это число переполнение) вызывается функцией term.

bool term(istream& in, int& res) – функция считающая произведение двух чисел возвращает в зависимости от полученного числа значение true или false (в зависимости от того, вызывает ли это число переполнение), вызывается функцией exp.

bool exp(istream& in, int& res) – функция считающая сумму и разность двух чисел возвращает в зависимости от полученного числа значение true или false (в зависимости от того, вызывает ли это число переполнение), вызывается main и exp.

**Экспериментальные результаты:** написанная программа должна остро реагировать на входящие и исходящие значения и должна выдавать Error1 при переполнении входящего значения и Error2 при переполнении исходящего значения. Ниже приведены некоторые из тестов:

|  |  |
| --- | --- |
| 111111111111+2\*3 | Error |
| 0\*0+0 | 0 |
| 2+99999\*999999+2 | Error |
| 150+23\*12-5 | 421 |
| 12345 | 12345 |

**Вывод:** было проведено ознакомление с материалом на тему «Рекурсия», способами реализации функции и наглядное отличие от итеративных функций.

#include <iostream>

#include <cmath>

using namespace std;

bool num(std::istream& in, int& res) {

in >> res;

if (!in.good()) cout << "Error\n";

return in.good();

}

bool term(std::istream& in, int& res) {

int val, vel;

if (!num(in, val))return false;

while (true) {

if (in.peek() == '\*') {

in.get();

if (!num(in, vel))return false;

if ((vel != 0) && (val > (INT\_MAX / vel))) {

cout << "Переполнение при умножении";

return false;

val \*= vel;

}

else {

res = val;

return true;

}

}

}

bool exp(std::istream& in, int& res) {

int val;

if(isdigit(in.peek())) if (!term(in, res))return false;

if (in.peek() == '+') {

in.get();

if (!term(in, val))return false;

if (res > (INT\_MAX - val)){

cout << "Переполнение при сложении";

return false;

}

res += val;

}

else if (in.peek() == '-') {

in.get();

if (!term(in, val))return false;

if (res < (INT\_MIN + val)){

cout << "Переполнение при вычитании";

return false;

}

res -= val;

}

else return true;

exp(in,res);

}

int main()

{

int res = 0;

exp(cin, res);

cout << res << endl;

system("pause > nul");

return 0;

}