

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Алгоритмы и структуры данных»
Тема: Рекурсия.

Студент гр. 7383

Левкович Д.В.

Преподаватель

Размочева Н.В.

Санкт-Петербург

2018

ОГЛАВЛЕНИЕ

Цель работы.....	3
Реализация задачи.....	4
Тестирование.....	6
Вывод.....	7
Приложение А. Тестовые случаи.....	8
Приложение Б. Исходный код программы.....	10

1. ЦЕЛЬ РАБОТЫ

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

Формулировка задачи: Написать программу, которая по заданному константному выражению вычисляем его значение либо сообщает о переполнении(превышении заданного значения) в процессе вычислений.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В функции `main` выводится приглашение выбрать способ ввода входных данных. После ввода данных, вызывается функция `pars(expr)`, которая вычисляет константное выражение. Если результат выражения больше, чем заданное значение, программа сообщает об этом и выводит. На каком шаге было переполнение.

Переменные, используемые в функции `main`:

- `expr` - строка, содержащая выражение.
- `Value` — переменная, в которой хранится результат.
- `k` — переменная для выбора ввода данных.

Лексема — минимальная, неделимая часть выражения.

Функция `pars` принимает указатель на строку с выражением и присваивает ей значение глобальной переменной `*expr`, чтобы все функции могли обращаться к этой строке. Далее функция вызывает другую функцию `getToken` и в переменную `token` попадает первая лексема. Потом вызывается первая из арифметических функций — `fSum` и ей передается адрес переменной `result`, в которой будет содержаться результат вычислений.

Функция `getToken` принимает указатель на обрабатываемую строку `*expr`. В начале функции объявлена статическая переменная `i`, которая хранит номер текущей позиции в анализируемой строке, далее происходит проверка, не достигнут ли конец строки, если он достигнут, то функция завершает работу. Потом с помощью функции `isspace` пропускаются все разделительные символы: пробел, табуляция. Далее с помощью функции `strchr` (поиск символа в строке), `isalpha`, `isdigit` происходит определение лексемы. Лексема записывается в глобальную переменную `token`. После этого функция возвращает адрес переменной `i` для того, чтобы можно было обнулить эту переменную вне функции `getToken`. После разбиения выражения на лексемы происходит его анализ. Точка входа в анализатор-функция `pars`.

Функция `fSum` позволяет вычислять сумму в выражении, также она

вызывает следующую функцию — `fMulti`, которая позволяет вычислять произведение в выражении. Функция `fMulti` в свою очередь вызывает функцию `fUnary`, которая позволяет вычислять, не является ли лексема унарным плюсом или унарным минусом, если это так, функция записывает лексему (+ или -) в переменную `op` и получает следующую лексему. Самой последней функцией в цепочке вызывается функция `fAtom`. Она с помощью функции `atoi` преобразует лексему в число типа `int` и записывает ее по адресу `*anw`, т. е. В переменную `result`.

Далее управление передается функциям в обратном порядке. После завершения `fAtom` управление переходит в функцию `fUnary`, она проверяет, содержится ли в переменной `op` унарный минус, если это так, она изменяет знак переменной `result`. Далее в функции `fMulti` происходит умножение. Вначале функция `fMulti` записывает оператор умножения в переменную `op`, затем вызывается функция `fUnary`, которой передается адрес переменной `temp` и вся цепочка начинается сначала. После завершения цепочки управление возвращается в функцию `fMulti`, а переменная `temp` содержит значение второго операнда для оператора, который сохранен в переменной `op`. После выполнения арифметических действий управление переходит к функции `fSum`, которая работает аналогично функции `fMulti`.

После всех действий, переменная `result` содержит результат вычислений. Функция `pars` копирует этот результат в исходную строку, на которую указывает `*exrg`, и работа анализатора заканчивается.

Если какая-либо функция обнаружила ошибку, она завершает свою работу вернув значение 1. Далее по цепочке завершаются все функции.

3. ТЕСТИРОВАНИЕ

Программа была собрана в компиляторе gcc в OS Linux Ubuntu 16.04
В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

4. ВЫВОД

В ходе работы были получены навыки рекурсивного программирования на языке С. Для работы использовался алгоритм рекурсивного спуска. Преимущество данного алгоритма заключается в том, что выполнение арифметических операций можно обеспечить в нужном порядке, в соответствии с законами математики. В каждой функции выполняются определенные арифметические действия, для выполнения которых требуется вызывать функции в нужной последовательности. Это и есть принцип алгоритма.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

Таблица 2 — Корректные тестовые случаи

Входные данные	Результат
10 1 1+2+3	6
1000 1 3*8+4*10+5*6*7*10	274
20 1 1+3*4	13
10 1 3*3+3	Переполнение
3	3

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>

int* getToken(char*); //Получает лексему из строки
void pars(char*); //Точка входа анализатора
int fSum(int*); //Обрабатывает сложение и вычитание
int fMulti(int*); //Обрабатывает умножение и деление
```



```

int fUnary(int*); //Обработка унарных операторов
int fAtom(int*); //Получает значение числа
int count = 0;
int max = 0;
char *expr; //Указатель на обрабатываемую строку
char token[80]; //Лексема

int main()
{
    int value=0;
    char expr[255]; //Содержит вычисляемое выражение
    int k = 0;
    FILE *in=fopen("input.txt", "r");
    FILE *out=fopen("output.txt", "w");
    printf("Введите максимальное значение результата выражения:")
    scanf("%d", &max);
    printf("Введите 1, если хотите ввести выражение из терминала, 2-из
файла:\n");
    scanf("%d\n", &k);
    switch(k){
    case 1:
        fgets(expr,80, stdin);
        if(!*expr) return 0; //Если введена пустая строка - завершить программу
        pars(expr); //Вычислить выражение
        value=atoi(expr);
        if(value>max){
            printf("Значение выражения(%d) больше максимума(%d). Программа
завершилась на %d-ом шаге.\n", value, max, count);
            return 0;
        }
        printf("Result: %d\n", value);
        break;
    case 2:
        fgets(expr, 80, in);
        pars(expr);
        value=atoi(expr);
        if(value>max){
            fprintf(out, "Значение выражения(%d) больше максимума(%d).
Программа завершилась на %d-ом шаге\n", value, max, count);
            return 0;
        }
        fprintf(out, "%d", value);
        break;
    default:

```

```

        printf("Введен неверный символ\n");
        break;

    }
    fclose(in);
    fclose(out);
    return 0;
}

void pars(char *line)
{
    int *pointer;
    int result;
    expr=line;
    pointer=getToken(expr);
    fSum(&result);
    *pointer=0;
    sprintf(expr, "%d", result);
}

int* getToken(char *expr)
{
    static int i=0;

    if(expr[i]=='\0') //Если конец выражения
    {
        i=0;
        return 0;
    }
    while(isspace(expr[i])) i++; //Пропустить разделительные символы

    if(strchr("+-*", expr[i]))
    {
        *token = expr[i];
        *(token+1) = '\0';
    }
    else if(isdigit(expr[i]))
    {
        int j=0;
        token[j]=expr[i];
        while(isdigit(expr[i+1]))
            token[++j]=expr[++i];
        token[j+1]='\0';
    }
    i++;
}

```

```

    return &i;
}

int fSum(int *anw)
{
    char op;
    int temp;
    if(fMulti(anw)) return 1;

    while((op = *token) == '+' || op == '-')
    {
        getToken(expr);
        fMulti(&temp);
        switch(op)
        {
            case '+':
                *anw += temp;
                count++;
                if(*anw>max)
                    return -1;
                break;
            case '-':
                *anw -= temp;
                count++;
                if(*anw>max)
                    return -1;
                break;
        }
    }

    return 0;
}

int fMulti(int *anw)
{
    char op;
    int temp;
    if(fUnary(anw)) return 1; //Ошибка

    while((op = *token) == '*')
    {
        getToken(expr);
        if(fUnary(&temp)) return 1; //Ошибка
        switch(op)
        {

```

```

        case '*':
            *anw *= temp;
            count++;
            if(*anw>max)
                return -1;
            break;
    }
}

return 0;
}

int fUnary(int *anw)
{
    char op=0;
    if(*token == '+' || *token == '-')
    {
        op = *token;
        getToken(expr);
    }
    if(fAtom(anw)) return 1; //Ошибка

    if(op == '-') *anw = -(*anw);

return 0;
}

int fAtom(int *anw){
    *anw = atoi(token);
    getToken(expr);
return 0;
}

```