

**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
1 10 1+2+3	6
1 1000 3*8+4*10+5*6*7*10	274
1 20 1+3*4	13
1 10 3*3+3	Переполнение
3	3
1+2+q	Syntax error
1++++ +	Syntax error

#### Некорректные случаи:

При тестировании программы возникла следующая ошибка — результат выражения «1+++» был равен «1», хотя это некорректное выражение. Для решения этой проблемы были добавлены перечисления с именованными константами «No» - ошибок нет, «Syntax» - ошибка синтаксиса. В строке данном некорректном выражении оператор «+» встречается несколько раз подряд, соответственно, анализатор не может его обработать. Все арифметические функции возвращают 0, если не встретилось



ошибок, иначе они возвращают 1. Если функция встретила ошибку, то она устанавливает необходимое значение ошибки в перечислении `errno` и затем все функции завершают свою работу, при этом возвращая 1.

## ПРИЛОЖЕНИЕ Б.

### ИСХОДНЫЙ КОД ПРОГРАММЫ

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

int* getToken(char*, char*); //Получает лексему из строки
void pars(char *expr, char* token, FILE* out, int* count, int* max, int* deep, int
k); //Точка входа анализатора
int fSum(int*,char*, FILE*, char *token, int *count, int *max, int *deep, int k);
//Обрабатывает сложение и вычитание
int fMulti(int*,char*, FILE*, char *token, int *count, int *max, int *deep, int
k); //Обрабатывает умножение и деление
int fUnary(int*, char* token, int* deep, char*); //Обработка унарных операторов
int fAtom(int*,char*, int* deep, char*); //Получает значение числа

enum {Empty, Operator, Number, Char} type;
enum {No, Syntax} error;

int main()
{
    int count = 0; //счетчик шагов
    int max = 0; //максимум выражения
    int deep = 0; //глубина
    char *expr=(char*)malloc(255*sizeof(char)); //Указатель на обрабатываемую
строку
    char token[80]; //Лексема
    int k = 0;
    int run=1;
    int value=0;
    //char expr[255]; //Содержит вычисляемое выражение
    FILE *in=fopen("input.txt", "r");
    FILE *out=fopen("output.txt", "w");
    while(run){
```

```

printf("Введите 1, если хотите ввести выражение из терминала, 2-из файла,
3-завершить программу: ");
scanf("%d", &k);
switch(k){
    case 1:
        printf("Введите максимальное значение результата выражения: ");
        scanf("%d", &max);
        getchar();
        printf("Введите выражение: ");
        fgets(expr, 255, stdin);
        if(!*expr)
            return 0; //Если введена пустая строка - завершить программу
        pars(expr, token, out, &count, &max, &deep, k); //Вычислить выражение
        value=atoi(expr);
        if(error==Syntax){
            printf("%s\n", expr);
        }
        if(value>max){
            printf("Значение выражения(%d) больше максимума(%d).
Программа завершилась на %d-ом шаге.\n", value, max, count);
            break;
        }
        printf("Result: %d, вложенность рекурсии %d\n\n", value, deep);
        break;
    case 2:
        fscanf(in, "%d\n", &max);
        fgets(expr, 80, in);
        fprintf(out, "%s", expr);
        pars(expr, token, out, &count, &max, &deep, k);
        value=atoi(expr);
        if(error==Syntax){
            fprintf(out, "%s\n", expr);
        }
        if(value>max){
            fprintf(out, "Значение выражения(%d) больше максимума(%d).
Программа завершилась на %d-ом шаге\n\n", value, max, count);
            break;
        }
        fprintf(out, "Result: %d, вложенность рекурсии %d\n\n", value, deep);

```

```

        break;
    case 3:
        return 0;
    default:
        printf("Введен неверный символ\n");
        break;
    }
    count=0;
    deep=0;
    if(k==3)
        run=0;
}
fclose(in);
fclose(out);
return 0;
}

```

```

void pars(char *expr,char* token, FILE* out, int* count,int* max, int* deep, int k)
{
    int *pointer;
    int result;
    error=No;
    pointer=getToken(expr, token);
    fSum(&result,expr, out, token, count, max, deep, k);
    *pointer=0;
    switch(error)
    {
        case No:
            sprintf(expr, "%d", result);
            break;
        case Syntax:
            strcpy(expr, "Syntax error!");
            break;
    }
}

```

```

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

```

```

type=Empty;
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';
    type=Operator;
}
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';
    type=Number;
}
else if(!isdigit(expr[i])){
    error=Syntax;
}
i++;
return &i;
}

```

```

int fSum(int *anw,char* expr, FILE* out, char* token, int* count, int* max, int*
deep, int k)
{
    char op;
    int temp;
    if(fMulti(anw,expr, out, token, count, max, deep, k)) return 1;
    while((op = *token) == '+' || op == '-')
    {
        getToken(expr, token);

```

```

fMulti(&temp,expr ,out, token, count, max, deep, k);
switch(op)
{
case '+':
if(k==1)
printf("Вы здесь: %d + %d\n", *anw, temp);
else if(k==2)
fprintf(out, "Вы здесь: %d + %d\n", *anw, temp);
*anw += temp;
(*count)++;
if(*anw>*max)
return -1;
break;
case '-':
if(k==1)
printf("Вы здесь: %d - %d\n", *anw, temp);
else if(k==2)
fprintf(out, "Вы здесь: %d - %d\n", *anw, temp);
*anw -= temp;
(*count)++;
if(*anw>*max)
return -1;
break;
}
}
(*deep)++;
return 0;
}

```

```

int fMulti(int *anw,char* expr, FILE* out, char* token, int* count, int* max, int*
deep, int k)
{
char op;
int temp;
if(fUnary(anw, token, deep, expr)) return 1; //Ошибка

while((op = *token) == '*')
{
getToken(expr, token);

```

```

    if(fUnary(&temp, token, deep, expr)) return 1; //Ошибка
    switch(op)
    {
        case '*':
            if(k==1)
                printf("Вы здесь: %d * %d\n", *anw, temp);
            else if(k==2)
                fprintf(out,"Вы здесь: %d * %d\n", *anw, temp);
                *anw *= temp;
                (*count)++;
                if(*anw>*max)
                    return -1;
                break;
    }
}
(*deep)++;
return 0;
}

```

```

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

    if(op == '-') *anw = -(*anw);
    (*deep)++;
    return 0;
}

```

```

int fAtom(int *anw,char* token, int* deep, char* expr){

    if(type==Number){
        *anw = atoi(token);
        getToken(expr, token);
    }
}

```

```
    }  
    else{  
        error=Syntax;  
        return 1;  
    }  
    (*deep)++;  
    return 0;  
}
```