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

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 *expr, 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 выполение арифметических операций можно обеспечить в нужном порядке, в соответствии с законами математики. В каждоый 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

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*, FILE*); //Точка входа анализатора
```

```
int fSum(int*, FILE*); //Обрабатывает сложение и вычитание
int fMulti(int*, FILE*); //Обрабатывает умножение и деление
int fUnary(int*); //Обработка унарных операторов
int fAtom(int*); //Получает значение числа

enum {Empty, Operator, Number} type;
int count = 0;//счетчик шагов
int max = 0;//максимум выражения
int deep = 0;//глубина
char *expr; //Указатель на обрабатываемую строку
char token[80]; //Лексема
int k = 0;

int main()
{
        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\n", &max);
                printf("Введите выражение:");
              fgets(expr,80, stdin);
              if(!*expr)
                return 0;    //Если введена пустая строка - завершить программу
              pars(expr, out); //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);
                 break;
              }
```

```
            printf("Result: %d, вложенность рекурсии %d\n", value, deep);
            break;
        case 2:
            fscanf(in,"%d\n",&max);
            fgets(expr, 80, in);
            fprintf(out,"%s", expr);
            pars(expr, out);
            value=atoi(expr);
            if(value>max){
                fprintf(out ,"Значение выражения(%d) больше максимума(%d).
Программа заверилась на %d-ом шаге\n\n", value, max, count);
                break;
            }
            fprintf(out, "Result: %d, вложенность рекурии %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 *line, FILE* out)
{
  int *pointer;
  int result;
  expr=line;
  pointer=getToken(expr);
  fSum(&result, out);
  *pointer=0;
```

```
    sprintf(expr, "%d", result);
}

int* getToken(char *expr)
{
    type=Empty;
    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';
        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;
    }
    i++;
return &i;
}

int fSum(int *anw, FILE* out)
{
    char op;
    int temp;
    if(fMulti(anw, out)) return 1;
```

```
   while((op = *token) == '+' || op == '-')
   {
      getToken(expr);
      fMulti(&temp, out);
      switch(op)
      {
       case '+':
       if(k==1)
        printf("Вы здесь: %d + %d\n", *anw, temp);
       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);
       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, FILE* out)
{
   char op;
   int temp;
   if(fUnary(anw)) return 1; //Ошибка

   while((op = *token) == '*')
```

```
    {
        getToken(expr);
        if(fUnary(&temp)) return 1; //Ошибка
        switch(op)
        {
         case '*':
         if(k==1)
          printf("Вы здесь: %d * %d\n", *anw, temp);
         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 op=0;
    if(*token == '+' || *token == '-')
    {
        op = *token;
        getToken(expr);
    }
    if(fAtom(anw)) return 1; //Ошибка

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

int fAtom(int *anw){

    if(type==Number){
```

```
        *anw = atoi(token);
        getToken(expr);
    }
    else
        return 1;
    deep++;
return 0;
}
```