ММИНОБРНАУКИ 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: Взаимно-рекурсивные функции и процедуры. Синтаксический анализатор понятия константного выражения.

Студент гр. 9384 _____ Мосин К.К.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Разработать алгоритм обработки строки, с помощью взаимно-рекурсивной функции, для нахождения “константного_выражения”, а также провести проверку на переполнение чисел типа int.

Задание.

ВАРИАНТ 11.

Построить синтаксический анализатор для определяемого далее понятия константное_выражение

1. константное_выражение::=ряд_цифр | константное_выражение знак_операции константное_выражение

3. знак_операции::=+ | - | *

4. ряд_цифр::=цифра | цифра ряд_цифр

Написать программу, которая по заданному (см. предыдущее задание) константному_выражению вычисляет его значение либо сообщает о переполнении (превышении заданного значения) в процессе вычислений.

Выполнение работы.

При запуске программы пользователю предложено ввести путь до тестируемых данных и путь до выходного файла. После чего построчно обрабатывается файл, введенный в качестве теста. Вызывается функция

bool expression(std::ofstream& outfile, char *str)

которая определяет достоверность тестируемой строки. Она должна соответствовать виду выражение_операция_выражение, где выражение соответствует выражение_операция_выражение или строка должна соответствовать числу. Функция

bool digit(std::ofstream& outfile, char *str, unsigned int& i)

перебирает посимвольно строку и проверяет не находятся ли в строке лишние символы. В удачном случае вызывается функция

bool exec(std::ofstream& outfile, char *str)

которая обрабатывает строку и выполняет операции сложения, вычитания и умножения слева направо и проверяет результат или же какой либо аргумент на переполнение. Все ошибки, полученные в результате выполнения программы обрабатываются вызовом функции

void error(std::ofstream& outfile, int error);

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные
71237172371239999998284832482848-238483284 123123123*123123123 qwjn3123 123123-12312 655-900 564*7 w21 18273-0	71237172371239999998284832482848-238483284 ERROR[5] - Операция вызывает переполнение 123123123*123123123 ERROR[5] - Операция вызывает переполнение qwjn3123 ERROR[2] - Выражение должно состоять только из цифр 123123-12312 = 110811 655-900 = -245 564*7 = 3948 w21 ERROR[2] - Выражение должно состоять только из цифр 18273-0 = 18273
12312*2 22* 231y-2 w2 3128-213 12w2+1 1+o	12312*2 = 24624 22* ERROR[4] - После знака операции следует число 231y-2 ERROR[2] - Выражение должно состоять только из цифр w2 ERROR[2] - Выражение должно состоять только из цифр 3128-213 = 2915 12w2+1 ERROR[2] - Выражение должно состоять только из цифр 1+o ERROR[4] - После знака операции следует число
w 13123 -213 123-12+2	ERROR[1] - Пустая строка w ERROR[2] - Выражение должно состоять только из цифр 13123 -213 ERROR[2] - Выражение должно состоять только из цифр

	123-12+2 = 113

Выводы.

В ходе выполнения лабораторный работы были изучены взаимно-рекурсивные функции и процедуры, а также, разработанная программа, выполняет синтаксический анализатор понятия константного_выражения.

ПРИЛОЖЕНИЕ
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <cctype>
#include <cstring>
#include <climits>
#include <cmath>

#define INIT 1000

bool expression(std::ofstream& outfile, char *str);
bool digit(std::ofstream& outfile, char *str, unsigned int& i);
void error(std::ofstream& outfile, int error);
bool exec(std::ofstream& outfile, char *str);
bool count(std::ofstream& outfile, char *num, char op, int& result, int index);
void reverse(char *str);

int main(){
   char input[INIT], output[INIT];

   std::cout << "Type input file path: ";
   std::cin >> input;
   std::ifstream infile(input);
   if (!infile){
      std::cout << "Входной файл не открыт!" << std::endl;
      return -1;
   }

   std::cout << "Type output file path: ";
   std::cin >> output;
```

```
    std::ofstream outfile(output);
    if (!outfile){
        std::cout << "Выходной файл не открыт!" << std::endl;
        return -1;
    }

    char str[INIT];
    while (!infile.eof()){
        infile.getline(str, INIT, '\n');
        str[strlen(str)] = '\0';
        outfile << str;
        bool exp = expression(outfile, str);
        if (exp){
            exec(outfile, str);
        }
    }
    return 0;
}

bool expression(std::ofstream& outfile, char *str){
    bool exp = false;
    unsigned int i = 0;
    if (str[i]){
        if (std::isdigit(str[i])){
            exp = digit(outfile, str, ++i);
        }
        else{
            error(outfile, 2);
            return false;
        }
```

```
        bool eof = str[i] == '\0' ? true : false;

        if (exp && !eof){
            error(outfile, 3);
            return false;
        }

        exp = (exp && eof);
    }
    else{
        error(outfile, 1);
        return false;
    }
    return exp;
}

bool digit(std::ofstream& outfile, char *str, unsigned int& i){
    if (str[i]){
        if (std::isdigit(str[i]) || ((str[i] == '+' || str[i] == '-' || str[i] == '*') && (str[i-1] != '+'
&& str[i-1] != '-' && str[i-1] != '*'))){
            return digit(outfile, str, ++i);
        }
        else if (str[i-1] == '+' || str[i-1] == '-' || str[i-1] == '*'){
            error(outfile, 4);
            return false;
        }
        else{
            error(outfile, 2);
            return false;
        }
    }
```

```
    else if (str[i-1] != '+' && str[i-1] != '-' && str[i-1] != '*'){
        return true;
    }
    else{
        error(outfile, 4);
        return false;
    }
}

bool exec(std::ofstream& outfile, char *str){
    unsigned int i = 0;
    unsigned int j = 1;
    unsigned int k = 0;
    unsigned int l = 0;
    unsigned int index = 0;
    char **numbers = new char*[INIT];
    char op[INIT];
    op[0] = '\0';
    op[1] = '?';
    char buffer[INIT];
    int result = 0;
    while (str[i]){
        if (str[i] == '+' || str[i] == '-' || str[i] == '*'){
            op[j++] = str[i++];
            buffer[k] = '\0';
            numbers[l] = new char[INIT];
            strcpy(numbers[l++], buffer);
            k = 0;
        }
        buffer[k++] = str[i++];
        if (str[i]=='\0'){
```

```
            numbers[l] = new char[INIT];
            buffer[k] = '\0';
            strcpy(numbers[l++], buffer);
            k = 0;
        }
    }
    if (op[1] == '?'){
        outfile << std::endl;
        return true;
    }
    for(unsigned int o = 0, p = 0; o < l; o++, p++){
        if (!count(outfile, numbers[o], op[p], result, index++)){
            error(outfile, 5);
            return false;
        }
    }
    outfile << " = " << result << std::endl;

    delete [] numbers;
    return true;
}

bool count(std::ofstream& outfile, char *num, char op, int& result, int index){
    char max[INIT];
    int maxNumber = INT_MAX;
    for (unsigned int i = 0; i < 10; i++){
        max[i] = (maxNumber % 10 + '0');
        maxNumber /= 10;
    }
    max[10] = '\0';
    reverse(max);
```

```
    if (strlen(num) > strlen(max) || (strlen(num) == strlen(max) && strcmp(num, max)
> 0)){
        return false;
    }

    if (index == 0){
        result = std::atoi(num);
        return true;
    }
    if (op == '+' && result < INT_MAX - std::atoi(num)){
        result += std::atoi(num);
    }
    else if (op == '-' && result > INT_MIN + std::atoi(num)){
        result -= std::atoi(num);
    }
    else if (op == '*' && result < INT_MAX / std::atoi(num)){
        result *= std::atoi(num);
    }
    else{
        return false;
    }

    return true;
}

void reverse(char *str){
    char string[INIT];
    for(unsigned int i = 0; i < strlen(str); i++){
        string[i] = str[strlen(str)-i-1];
    }
    strcpy(str, string);
```

```
}

void error(std::ofstream& outfile, int error){
   switch (error){
   case 1:
      outfile << " ERROR[" << error << "] - Пустая строка" << std::endl;
      break;

   case 2:
      outfile << " ERROR[" << error << "] - Выражение должно состоять только из цифр" << std::endl;
      break;

   case 3:
      outfile << " ERROR[" << error << "] - Выражение содержит лишние символы" << std::endl;
      break;

   case 4:
      outfile << " ERROR[" << error << "] - После знака операции следует число" << std::endl;
      break;

   case 5:
      outfile << " ERROR[" << error << "] - Операция вызывает переполнение" << std::endl;
      break;

   default:
      outfile << " ?" << std::endl;
   }
}
```