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. 9381 Матвеев А. Н.

Преподаватель Фирсов М. А.

Санкт-Петербург
2020

Цель работы.

Понять принцип рекурсии, научиться работать с рекурсивными и взаимно-рекурсивными функциями, написать программу на языке C++, вычисляющую значение логического выражения.

Задание.

Вариант 9.

Разработать программу, которая по заданному простому_логическому выражению (определение понятия см. в предыдущей задаче), не содержащему вхождений простых идентификаторов, вычисляет значение этого выражения.

простое_логическое::= TRUE | FALSE | *простой_идентификатор* |
NOT *простое_логическое* |
(простое_логическое знак_операции простое_логическое)
простой-идентификатор::=буква
знак-операции::= AND | OR

Уточнение задания.

Для удобства слова интерпретированы в символы, а именно:
TRUE = 1; FALSE = 0; AND = &; OR = | ; NOT = ~.

Выполнение работы и описание алгоритма.

Описание алгоритма:

Используется метод нисходящего рекурсивного спуска и идея состоит в применении форм Бэкуса-Наура - формальной системы описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие категории:

```
E -> T + E | T
T -> F * T | F
F -> N     | (E)
```

(В качестве нижнего уровня используются сразу две функции - fact и nfact, о которых будет сказано далее. Они весьма похожи по своей реализации,

однако fact работает с выражением, перед которым нет унарного минуса, а nfact - с выражением, в котором он есть.)

Используется перечисление:

enum attribs { EOI, NUM, OR, AND, NOT, LP, RP } - для возможности использовать оператор switch(), упрощающий структуру кода.

Используется структура:

```
typedef struct
{
   enum attribs attrib;
   char symbol;
} symbol_map;
```

Она упрощает работу символами входной строки, позволяет абстрагироваться от символов и рассматривать их в операторе switch() через их атрибуты, что удобно при создании масштабной программы. symbol - это текущий символ, который будет анализироваться алгоритмом.

int interpreter(const char* str); - функция-интерпретатор, позволяет преобразовывать слова в необходимые символы. Каждому слову соответствует некоторый символ, который и возвращает данная функция. Принимает на вход некоторое слово, которое нужно будет интерпретировать в символ.

Также имеется пространство имён main_vars, в котором объявлены переменные:

unsigned int str_index; - переменная, обозначающая текущий индекс входной строки.

char* input_str = NULL; - указатель на входную строку.

symbol_map get_next_symbol(void); - устанавливает текущему символу соответствующий атрибут и в случае успешного считывания сдвигает индекс входной строки на 1 вперёд.

void print_indent(int n, ofstream& fout) - печатает необходимое количество отступов при отладочных выводах в файл, чтобы визуализировать идею рекурсии.

Все нижеприведённые функции принимают в качестве второго аргумента число, позволяющее регулировать отступы, а в качестве третьего - ссылку на поток вывода - это необходимо для возможности выводить информацию в файл.

В качестве первого аргумента все они принимают ссылку на экземпляр структуры symbol_map, обозначающую текущий символ и его синтаксический атрибут (число (1|0) | левая скобка | правая скобка | логическое “и” | логическое “или” | логическое “не”) .

string statement(char* str, int n, ofstream &fout); - принимает на вход интерпретированную в набор символов строку, устанавливает глобальную переменную-указатель в начало строки и начальное значение индекса. После этого считывается следующий символ из входной строки при помощи get_next_symbol. Далее в операторе switch() устанавливается атрибут этого символа и вызывается функция expr. Возвращаемое ей значение присваивается переменной result. Она имеет либо нулевое, либо ненулевое значение. В зависимости от этого функция вернёт строку типа string “TRUE” или “FALSE”.

int expr(symbol_map &symb, int n, ofstream &fout); - функция, играющая роль верхней синтаксической категории. Она разбирает текущий символ: если он - число, левая скобка или “не”, тогда будущему возвращаемому значению присваивается результат работы функции term, после чего считывается следующий символ. Далее запускается switch(), в котором анализируется этот символ: если он - “или” , то считывается следующий, после чего результат суммируется с возвращаемым значением этой же самой функции. Если он - закрывающая скобка, то возвращаем его назад в строку путём декрементирования str_index. Функция возвращает целое число типа int.

int term(symbol_map &symb, int n, ofstream &fout); функция, обозначающая среднюю синтаксическую категорию. Рассматривает текущий

символ: если его атрибут - левая скобка или число, то вызывается функция fact и индекс сдвигается на следующий символ, после чего если он - логическое “и”, то считывается следующий и возвращаемый функцией результат умножается на очередной вызов этой же функции с только что считанным символом. В случае логического “или” или закрывающей скобки индекс смещается влево.

Если же исходный текущий символ имеет атрибут NOT, то происходят аналогичные действия, только вместо функции fact вызывается nfact, предназначенная для работы с отрицанием.

Функция int fact(symbol_map &symb, int n, ofstream &fout) - нижняя синтаксическая категория. Как и в остальных функциях, создаётся целочисленное значение ret, которое будет возвращено из функции, изначально равное 0. Если текущий символ - число, то оно присваивается переменной ret и функция завершается. Если открывающая скобка, то считывается следующий символ и переменной ret присваивается значение, возвращаемое функцией expr , после чего происходит ещё одно считывание.

Функция int nfact(symbol_map &symb, int n, ofstream &fout) - нижняя синтаксическая категория, наравне с fact. Считывается символ. Если он - число, то переменной ret присваивается число, обратное данному с точки зрения 0 | не 0. Если это левая скобка, то всё происходит аналогично функции fact, только переменной ret присваивается обратное возвращаемому значению функцией expr. Если текущий символ имеет атрибут NOT, то переменной ret присваивается обратное значение возвращаемому функцией nfact.

Во всех случаях по умолчанию функции прекращают работу.

В функции int main() открываются потоки ввода и вывода в файл из файла. Пользователь вводит пути источника и назначения. Далее считывается строка при помощи функции getline(). Далее строка разбивается на слова благодаря функции strtok из библиотеки cstring, а те, в свою очередь, на символы при помощи функции interpreter(). Далее создаётся переменная result типа string, куда записывается результат работы функции statement, являющейся некой стартовой точкой. Все отладочные выводы записываются в

файл, результат записывается в файл и выводится в консоль.

Если кратко описывать весь алгоритм работы, можно сказать, что expr анализирует выражения в скобках между которыми стоит “ИЛИ”. Функция term анализирует выражения в скобках между которыми стоит “И”, fact анализирует либо тривиальные значения, либо ЛЮБЫЕ выражения в скобках. nfact - любые тривиальные значения и выражения в скобках, перед которыми стоит “НЕ”. Все эти функции имеют доступ к конкретному символу и, проанализировав его, последовательно передают управление друг другу. На выходе получается конечное логическое значение.

Исходный код программы смотреть в Приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	TRUE AND NOT FALSE	Зашёл в функцию statement. Аргумент: 1&~0. Глубина рекурсии = 0 Зашёл в функцию expr. Текущий символ: 1. Глубина рекурсии = 1 Зашёл в функцию term. Текущий символ: 1. Глубина рекурсии = 2 Зашёл в функцию fact. Текущий символ: 1. Глубина рекурсии = 3 Вышел из функции fact. Возвращаемое значение: 1 Зашёл в функцию term. Текущий символ: ~. Глубина рекурсии = 3 Зашёл в функцию nfact. Текущий символ: ~. Глубина рекурсии = 4 Вышел из функции nfact. Возвращаемое значение: 1 Вышел из функции term. Возвращаемое значение: 1

		Вышел из функции term. Возвращаемое значение: 1 Вышел из функции expr. Возвращаемое значение: 1 Вышел из функции statement. Возвращаемое значение: TRUE Значение выражения: TRUE
2.	NOT (TRUE AND NOT FALSE)	Зашёл в функцию statement. Аргумент: ~(1&~0). Глубина рекурсии = 0 Зашёл в функцию expr. Текущий символ: ~. Глубина рекурсии = 1 Зашёл в функцию term. Текущий символ: ~. Глубина рекурсии = 2 Зашёл в функцию nfact. Текущий символ: ~. Глубина рекурсии = 3 Зашёл в функцию expr. Текущий символ: 1. Глубина рекурсии = 4 Зашёл в функцию term. Текущий символ: 1. Глубина рекурсии = 5 Зашёл в функцию fact. Текущий символ: 1. Глубина рекурсии = 6 Вышел из функции fact. Возвращаемое значение: 1 Зашёл в функцию term. Текущий символ: ~. Глубина рекурсии = 6 Зашёл в функцию nfact. Текущий символ: ~. Глубина рекурсии = 7 Вышел из функции nfact. Возвращаемое значение: 1 Вышел из функции term. Возвращаемое значение: 1

		Вышел из функции term. Возвращаемое значение: 1 Вышел из функции expr. Возвращаемое значение: 1 Вышел из функции nfact. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Вышел из функции expr. Возвращаемое значение: 0 Вышел из функции statement. Возвращаемое значение: FALSE Значение выражения: FALSE
3.	TRUE AND (FALSE OR (NOT FALSE AND (TRUE AND FALSE)))	Зашёл в функцию statement. Аргумент: 1&(0|(~0&(1&0))). Глубина рекурсии = 0 Зашёл в функцию expr. Текущий символ: 1. Глубина рекурсии = 1 Зашёл в функцию term. Текущий символ: 1. Глубина рекурсии = 2 Зашёл в функцию fact. Текущий символ: 1. Глубина рекурсии = 3 Вышел из функции fact. Возвращаемое значение: 1 Зашёл в функцию term. Текущий символ: (. Глубина рекурсии = 3 Зашёл в функцию fact. Текущий символ: (. Глубина рекурсии = 4 Зашёл в функцию expr. Текущий символ: 0. Глубина рекурсии = 5 Зашёл в функцию term. Текущий символ: 0. Глубина рекурсии = 6

		Зашёл в функцию fact. Текущий символ: 0. Глубина рекурсии = 7 Вышел из функции fact. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Зашёл в функцию expr. Текущий символ: (. Глубина рекурсии = 6 Зашёл в функцию term. Текущий символ: (. Глубина рекурсии = 7 Зашёл в функцию fact. Текущий символ: (. Глубина рекурсии = 8 Зашёл в функцию expr. Текущий символ: ~. Глубина рекурсии = 9 Зашёл в функцию term. Текущий символ: ~. Глубина рекурсии = 10 Зашёл в функцию nfact. Текущий символ: ~. Глубина рекурсии = 11 Вышел из функции nfact. Возвращаемое значение: 1 Зашёл в функцию term. Текущий символ: (. Глубина рекурсии = 11 Зашёл в функцию fact. Текущий символ: (. Глубина рекурсии = 12

		Зашёл в функцию expr. Текущий символ: 1. Глубина рекурсии = 13 Зашёл в функцию term. Текущий символ: 1. Глубина рекурсии = 14 Зашёл в функцию fact. Текущий символ: 1. Глубина рекурсии = 15 Вышел из функции fact. Возвращаемое значение: 1 Зашёл в функцию term. Текущий символ: 0. Глубина рекурсии = 15 Зашёл в функцию fact. Текущий символ: 0. Глубина рекурсии = 16 Вышел из функции fact. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Вышел из функции expr. Возвращаемое значение: 0

		Вышел из функции fact. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Вышел из функции expr. Возвращаемое значение: 0 Вышел из функции fact. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Вышел из функции expr. Возвращаемое значение: 0 Вышел из функции expr. Возвращаемое значение: 0 Вышел из функции fact. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Вышел из функции expr. Возвращаемое значение: 0 Вышел из функции statement. Возвращаемое значение: FALSE Значение выражения: FALSE
4.	NOT NOT FALSE	Зашёл в функцию statement. Аргумент: ~~0. Глубина рекурсии = 0

		Зашёл в функцию expr. Текущий символ: ~. Глубина рекурсии = 1 Зашёл в функцию term. Текущий символ: ~. Глубина рекурсии = 2 Зашёл в функцию nfact. Текущий символ: ~. Глубина рекурсии = 3 Зашёл в функцию nfact. Текущий символ: ~. Глубина рекурсии = 4 Вышел из функции nfact. Возвращаемое значение: 1 Вышел из функции nfact. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Вышел из функции expr. Возвращаемое значение: 0 Вышел из функции statement. Возвращаемое значение: FALSE Значение выражения: FALSE
5.	(TRUE AND (NOT TRUE OR TRUE))	Зашёл в функцию statement. Аргумент: (1&(~1|1)). Глубина рекурсии = 0 Зашёл в функцию expr. Текущий символ: (. Глубина рекурсии = 1 Зашёл в функцию term. Текущий символ: (. Глубина рекурсии = 2 Зашёл в функцию fact. Текущий символ: (. Глубина рекурсии = 3 Зашёл в функцию expr. Текущий символ: 1. Глубина рекурсии = 4 Зашёл в функцию term. Текущий символ: 1. Глубина рекурсии = 5 Зашёл в функцию fact. Текущий символ: 1. Глубина рекурсии = 6

		Вышел из функции fact. Возвращаемое значение: 1 Зашёл в функцию term. Текущий символ: (. Глубина рекурсии = 6 Зашёл в функцию fact. Текущий символ: (. Глубина рекурсии = 7 Зашёл в функцию expr. Текущий символ: ~. Глубина рекурсии = 8 Зашёл в функцию term. Текущий символ: ~. Глубина рекурсии = 9 Зашёл в функцию nfact. Текущий символ: ~. Глубина рекурсии = 10 Вышел из функции nfact. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Зашёл в функцию expr. Текущий символ: 1. Глубина рекурсии = 9 Зашёл в функцию term. Текущий символ: 1. Глубина рекурсии = 10 Зашёл в функцию fact. Текущий символ: 1. Глубина рекурсии = 11 Вышел из функции fact. Возвращаемое значение: 1

		Вышел из функции term. Возвращаемое значение: 1 Вышел из функции expr. Возвращаемое значение: 1 Вышел из функции expr. Возвращаемое значение: 1 Вышел из функции fact. Возвращаемое значение: 1 Вышел из функции term. Возвращаемое значение: 1 Вышел из функции term. Возвращаемое значение: 1 Вышел из функции expr. Возвращаемое значение: 1 Вышел из функции fact. Возвращаемое значение: 1 Вышел из функции term. Возвращаемое значение: 1 Вышел из функции expr. Возвращаемое значение: 1 Вышел из функции statement. Возвращаемое значение: TRUE Значение выражения: TRUE
6.	(TRUE AND FALSE)	Зашёл в функцию statement. Аргумент: (1&0). Глубина рекурсии = 0 Зашёл в функцию expr. Текущий символ: (. Глубина рекурсии = 1 Зашёл в функцию term. Текущий символ: (. Глубина рекурсии = 2 Зашёл в функцию fact. Текущий символ: (. Глубина рекурсии = 3

		Зашёл в функцию expr. Текущий символ: 1. Глубина рекурсии = 4 Зашёл в функцию term. Текущий символ: 1. Глубина рекурсии = 5 Зашёл в функцию fact. Текущий символ: 1. Глубина рекурсии = 6 Вышел из функции fact. Возвращаемое значение: 1 Зашёл в функцию term. Текущий символ: 0. Глубина рекурсии = 6 Зашёл в функцию fact. Текущий символ: 0. Глубина рекурсии = 7 Вышел из функции fact. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Вышел из функции expr. Возвращаемое значение: 0 Вышел из функции fact. Возвращаемое значение: 0 Вышел из функции term. Возвращаемое значение: 0 Вышел из функции expr. Возвращаемое значение: 0 Вышел из функции statement. Возвращаемое значение: FALSE Значение выражения: FALSE
7.	TRUE	Зашёл в функцию statement. Аргумент: 1. Глубина рекурсии = 0

		Зашёл в функцию expr. Текущий символ: 1. Глубина рекурсии = 1 Зашёл в функцию term. Текущий символ: 1. Глубина рекурсии = 2 Зашёл в функцию fact. Текущий символ: 1. Глубина рекурсии = 3 Вышел из функции fact. Возвращаемое значение: 1 Вышел из функции term. Возвращаемое значение: 1 Вышел из функции expr. Возвращаемое значение: 1 Вышел из функции statement. Возвращаемое значение: TRUE Значение выражения: TRUE

Выводы.

Написана программа, вычисляющая значение логического выражения методом рекурсивного спуска. Изучен принцип рекурсии и получены навыки в алгоритмах парсинга.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include<cstring>
#include <fstream>
#include <cstdlib>

using namespace std;
enum attribs { EOI, NUM, OR, AND, NOT, LP, RP };
typedef struct
{
    enum attribs attrib;
    char symbol;
} symbol_map;

string statement(char* input, int n, ofstream &fout);
int expr(symbol_map &symb, int n, ofstream &fout);
int term(symbol_map &symb, int n, ofstream &fout);
int fact(symbol_map &symb, int n, ofstream &fout);
int nfact(symbol_map &symb, int n, ofstream &fout);
char interpreter(const char* str);
symbol_map get_next_symbol();
namespace main_vars {
    unsigned int str_index;
    char *input_str = nullptr;
}
int main() { /* головная функция, тут реализован ввод и вывод их файла,
преобразование строки из слов в
       в символьную и вызов функции statement, которая является точкой запуска
рекурсивного алгоритма */
    ifstream fin;
    ofstream fout;
    string path1, path2;
    cout << "Введите путь источника и путь назначения:\n";
    cin >> path1 >> path2;
    fin.open(path1);
    fout.open(path2);
    if(fin.is_open()) cout << "Файл " << path1<< " успешно открыт\n";
    else{
        cout << "Не удалось открыть файл " << path2;
```

```
        return 0;
    }
    string input;
    getline(fin, input);
    fin.close();
    char *pch;
    int len = input.size();
    int count = 0;
    int str_size =0;
    for(int i = 0; i < len; i++) {
        if (input.substr(i, 1) == " ")
            str_size++;
    }
    str_size+=2;
    char* input_str_cpy = new char[len+1];
    strcpy(input_str_cpy, input.c_str());
    char *str = new char[str_size];
    pch = strtok(input_str_cpy, " ");
    while(pch){
        str[count] = interpreter(pch);
        count++;
        pch = strtok(nullptr, " ");
    }
    str[count] = '\0';
    string result = statement(str, 0, fout);
    cout << "Значение выражения: " << result <<"\n";
    fout << "Значение выражения: " << result <<"\n";
    delete [] str;
    delete [] input_str_cpy;
    fout.close();
    return 0;
}
void print_indent(int n, ofstream& fout){
    for(int i = 0; i < n; i++)
        fout << "  ";
}
```

```
string statement(char* input, int n, ofstream &fout) { /* анализируется первый символ строки, после чего
 * запускается алгоритм рекурсивного нисходящего спуска и вычисляется результат выражения */
```

```
    print_indent(n, fout);
     fout << "Зашёл в функцию statement. " << "Аргумент: " << input << ". Глубина
рекурсии = " << n << "\n";
    int result = 0;
    symbol_map symb;
    main_vars::input_str = input;
    main_vars::str_index = 0;
    symb = get_next_symbol();
    switch (symb.attrib) {
        case LP:
        case NUM:
        case NOT:
            // expr может начинаться с открывающейся скобки,  числа или слова not
            result = expr(symb, n + 1, fout);
            break;
        default:
            break;
    }
    print_indent(n, fout);
    fout << "Вышел из функции statement. Возвращаемое значение: ";
    if (result > 0) {
        fout << "TRUE\n";
        return "TRUE";
    } else {
        fout << "FALSE\n";
        return "FALSE";
    }
}

int expr(symbol_map &symb, int n, ofstream &fout) { // разбирает значения, между
которыми стоит or
    print_indent(n, fout);
     fout << "Зашёл в функцию expr. " << "Текущий символ: " << symb.symbol << ".
Глубина рекурсии = " << n << "\n";
    symbol_map tmp_symb = symb;
    int ret = 0;
    switch (tmp_symb.attrib) {
        case NUM:
        case LP:
        case NOT:
            ret = term(tmp_symb, n + 1, fout);
            tmp_symb = get_next_symbol();
            switch (tmp_symb.attrib) {
```

```
                case OR:
                    tmp_symb = get_next_symbol();
                    ret += expr(tmp_symb, n + 1, fout);
                    break;
                case RP:
                        // Если символ равен закрывающейся скобке, возвращаем его
назад в строку
                   main_vars:: str_index--;
                default:
                    break;
            }
            break;
        default:
            break;
    }
    print_indent(n, fout);
    fout << "Вышел из функции expr. Возвращаемое значение: " << ret << "\n";
    return ret;
}

int term(symbol_map &symb, int n, ofstream &fout) { // разбирает выражения, между
которыми стоит and
    print_indent(n, fout);
     fout << "Зашёл в функцию term. " << "Текущий символ: " << symb.symbol << ".
Глубина рекурсии = " << n << "\n";
    symbol_map tmp_symb = symb;
    int ret = 0;
    switch (tmp_symb.attrib) {
        case LP:
        case NUM:
            ret = fact(tmp_symb, n + 1, fout);
            tmp_symb = get_next_symbol();
            switch (tmp_symb.attrib) {
                case AND:
                    tmp_symb = get_next_symbol();
                    ret *= term(tmp_symb, n + 1, fout);
                    break;
                case OR:
                case RP:
                            // Если символ равен плюсу или закрывающейся скобке,
возвращаем его назад
                    main_vars:: str_index--;
                default:
```

```
                    break;
            }
            break;
        case NOT:
            ret = nfact(symb, n + 1, fout);
            tmp_symb = get_next_symbol();
            switch (tmp_symb.attrib) {
                case AND:
                    tmp_symb = get_next_symbol();
                    ret *= term(tmp_symb, n + 1, fout);
                    break;
                case OR:
                case RP:
                        // Если текущий символ равен or или закрывающейся скобке,
возвращаем его назад
                    main_vars:: str_index--;
                default:
                    break;
            }
            break;

        default:
            break;
    }
    print_indent(n, fout);
    fout << "Вышел из функции term. Возвращаемое значение: " << ret << "\n";
    return ret;
}
int nfact(symbol_map &symb, int n, ofstream &fout) {
     // разбирает разбирает  любые элементарные значения  и выражения в скобках,
если перед ними стоит отрицание
    print_indent(n, fout);
     fout << "Зашёл в функцию nfact. " << "Текущий символ: " << symb.symbol << ".
Глубина рекурсии = " << n << "\n";
    int ret = 0;
    symbol_map tmp_symb = get_next_symbol();
    switch (tmp_symb.attrib) {
        case NUM:
            ret = (atoi(&tmp_symb.symbol) == 0);
            break;
        case LP:
            tmp_symb = get_next_symbol();
            ret = (expr(tmp_symb, n + 1, fout) == 0);
```

```
            tmp_symb = get_next_symbol();
            break;
        case NOT:
            ret = !nfact(symb, n + 1, fout);
            break;
        default:
            break;
    }
    print_indent(n, fout);
    fout << "Вышел из функции nfact. Возвращаемое значение: " << ret << "\n";
    return ret;
}
int fact(symbol_map &symb, int n, ofstream &fout) { // разбирает элементарные значения и выражения в скобках
    print_indent(n, fout);
     fout << "Зашёл в функцию fact. " << "Текущий символ: " << symb.symbol << ". Глубина рекурсии = " << n << "\n";
    int ret = 0;
    symbol_map tmp_symb = symb;
    switch (tmp_symb.attrib) {
        case NUM:
            ret = atoi(&tmp_symb.symbol);
            break;
        case LP:

            tmp_symb = get_next_symbol();
            ret = expr(tmp_symb, n + 1, fout);
            tmp_symb = get_next_symbol();
            // Считываем закрывающуюся скобку
            break;
        default:
            break;
    }
    print_indent(n, fout);
    fout << "Вышел из функции fact. Возвращаемое значение: " << ret << "\n";
    return ret;
}

using namespace main_vars;

symbol_map get_next_symbol() /* последовательно даёт доступ к символам логического выражения ,
 * присваивает им атрибуты. За каждый запуск символ сдвигается на 1 */
```

```
{
    symbol_map cur_symb;
    cur_symb.attrib = EOI;

    switch (input_str[str_index])
    {
        case '0':
        case '1':
            cur_symb.attrib = NUM;
            break;
        case '|':
            cur_symb.attrib = OR;
            break;
        case '&':
            cur_symb.attrib = AND;
            break;
        case '(':
            cur_symb.attrib = LP;
            break;
        case ')':
            cur_symb.attrib = RP;
            break;
        case '~':
            cur_symb.attrib = NOT;
            break;
        default:
            break;
    }

    if (cur_symb.attrib != EOI)
    {
        cur_symb.symbol = input_str[str_index];
        str_index++;
    }

    return cur_symb;
}

char interpreter(const char* str){ // позволяет преобразовать строку, состоящую из
слов, в символьную
    switch(*str){
        case 'T':
            return '1';
```

```
        case 'F':
            return '0';
        case 'A':
            return '&';
        case 'O':
            return '|';
        case '(':
            return '(';
        case ')':
            return ')';
        case 'N':
            return '~';
        default:
            return '*';
    }
}
```