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. 9382 Иерусалимов 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:
NOT = ‘ ` ’ ; AND = ‘&’; OR = ‘|’; TRUE = ‘1’; FALSE = ‘0’;

Выполнение работы и описание алгоритма.

Описание алгоритма:

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

Присутствует перечисления оно упрощает работу с символами входной строки, позволяет абстрагироваться от символов:

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;
```

Далее через оператор switch() рассматриваем их атрибуты и выполняется нужное действие.

symbol - это текущий символ, который будет анализироваться алгоритмом.

char interpretator(const char* str); - функция-интерпретатор, принимает на вход слово которое интерпретирует его в символ и возвращает назад.

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 getNext(); - функция устанавливает соответствующий атрибут символу и в случае удачного считывания сдвигает индекс текущего символа на единицу для последующих действий.

void printDepth(int n, ofstream& fout) – печатает нужно количество точек друг за другом чтобы визуализировать рекурсию.

Все нижеприведённые функции принимают в качестве второго аргумента число, которое позволяет регулировать отступы для функции void printDepth(int n, ofstream& fout), а в качестве третьего аргументы ссылку на потоковый ввод 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 state(char* str, int n, ofstream &fout); - принимает на вход набор символов , устанавливает глобальную переменную-указатель в начало строки и начальное значение индекса. После этого считывается следующий символ из входной строки при помощи getNext(). Далее в операторе switch() устанавливается атрибут этого символа и вызывается функция orExept. 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 orExept(symbolFieald &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 parentheses 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 parentheses вызывается nparentheses, предназначенная для работы с отрицанием.

Функция int parentheses (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 nparentheses(symbol_map &symb, int n, ofstream &fout) - нижняя синтаксическая категория, наравне с parentheses. 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 parentheses, только переменной ret присваивается обратное возвращаемому значению функцией orExept. Если текущий символ имеет атрибут NOT, то переменной ret присваивается обратное значение возвращаемому функцией nparentheses.

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 interpretator(). Далее создаётся переменная result типа string, куда записывается результат работы функции state, 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 orExept анализирует выражения в скобках между которыми стоит “ИЛИ”. Функция term анализирует выражения в скобках между которыми стоит “И”, parentheses анализирует либо тривиальные значения, либо ЛЮБЫЕ выражения в скобках. nparentheses - 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.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№	Входные данные	Выходные данные
1.	TRUE	Зашёл в функцию state. Аргумент: 1 .Зашёл в функцию orExept. Текущий символ: 1 ..Зашёл в функцию term. Текущий символ: 1 ...Зашёл в функцию parentheses. Текущий символ: 1 ...Вышел из функции parentheses. Возвращаемое значение: 1 ..Вышел из функции term. Возвращаемое значение: 1 .Вышел из функции orExept. Возвращаемое значение: 1 Вышел из функции state. Возвращаемое значение: TRUE Значение выражения: TRUE
2.	FALSE AND TRUE	Зашёл в функцию state. Аргумент: 0&1 .Зашёл в функцию orExept. Текущий символ: 0 ..Зашёл в функцию term. Текущий символ: 0 ...Зашёл в функцию parentheses. Текущий символ: 0 ...Вышел из функции parentheses. Возвращаемое значение: 0 ...Зашёл в функцию term. Текущий символ: 1 ....Зашёл в функцию parentheses. Текущий символ: 1 ....Вышел из функции parentheses. Возвращаемое значение: 1 ...Вышел из функции term. Возвращаемое значение: 1 ..Вышел из функции term. Возвращаемое значение: 0 .Вышел из функции orExept. Возвращаемое значение: 0 Вышел из функции state. Возвращаемое значение: FALSE Значение выражения: FALSE
3.	NOT (TRUE AND FALSE)	Зашёл в функцию state. Аргумент: `(&0

		.Зашёл в функцию orExept. Текущий символ: ` ..Зашёл в функцию term. Текущий символ: ` ...Зашёл в функцию nparentheses. Текущий символ: ` ....Зашёл в функцию orExept. Текущий символ: & ....Вышел из функции orExept. Возвращаемое значение: 0 ...Вышел из функции nparentheses. Возвращаемое значение: 1 ..Вышел из функции term. Возвращаемое значение: 1 .Вышел из функции orExept. Возвращаемое значение: 1 Вышел из функции state. Возвращаемое значение: TRUE Значение выражения: TRUE
4.	(TRUE AND (FALSE OR (TRUE AND TRUE)))	Зашёл в функцию state. Аргумент: (1&(0|(1&1))) .Зашёл в функцию orExept. Текущий символ: (..Зашёл в функцию term. Текущий символ: (...Зашёл в функцию parentheses. Текущий символ: (....Зашёл в функцию orExept. Текущий символ: 1 .....Зашёл в функцию term. Текущий символ: 1 ......Зашёл в функцию parentheses. Текущий символ: 1 ......Вышел из функции parentheses. Возвращаемое значение: 1 ......Зашёл в функцию term. Текущий символ: (.......Зашёл в функцию parentheses. Текущий символ: (........Зашёл в функцию orExept. Текущий символ: 0 ....Зашёл в функцию term. Текущий символ: 0 ....Зашёл в функцию parentheses. Текущий символ: 0 ....Вышел из функции parentheses. Возвращаемое значение: 0 ....Вышел из функции term. Возвращаемое значение: 0 ....Зашёл в функцию orExept. Текущий символ: (..........Зашёл в функцию term. Текущий символ: (...........Зашёл в функцию parentheses. Текущий символ: (............Зашёл в функцию orExept. Текущий символ: 1 ....Зашёл в функцию term. Текущий символ: 1 ....Зашёл в функцию parentheses. Текущий символ: 1

		.....Вышел из функции parentheses. Возвращаемое значение: 1 .....Зашёл в функцию term. Текущий символ: 1 .....Зашёл в функцию parentheses. Текущий символ: 1 .....Вышел из функции parentheses. Возвращаемое значение: 1 .....Вышел из функции term. Возвращаемое значение: 1 .....Вышел из функции term. Возвращаемое значение: 1 .....Вышел из функции orExept. Возвращаемое значение: 1 .....Вышел из функции parentheses. Возвращаемое значение: 1 .....Вышел из функции term. Возвращаемое значение: 1 .....Вышел из функции orExept. Возвращаемое значение: 1 .....Вышел из функции orExept. Возвращаемое значение: 1 .....Вышел из функции parentheses. Возвращаемое значение: 1 .....Вышел из функции term. Возвращаемое значение: 1 .....Вышел из функции term. Возвращаемое значение: 1 .....Вышел из функции orExept. Возвращаемое значение: 1 ...Вышел из функции parentheses. Возвращаемое значение: 1 ..Вышел из функции term. Возвращаемое значение: 1 .Вышел из функции orExept. Возвращаемое значение: 1 Вышел из функции state. Возвращаемое значение: TRUE Значение выражения: TRUE
5.	NOT (((TRUE OR TRUE) AND (FALSE OR TRUE)) AND TRUE)	Зашёл в функцию state. Аргумент: `(((1|1)&(0|1)&1 .Зашёл в функцию orExept. Текущий символ: ` ..Зашёл в функцию term. Текущий символ: `

		...Зашёл в функцию nparentheses. Текущий символ: ` ....Зашёл в функцию orExept. Текущий символ: (.....Зашёл в функцию term. Текущий символ: (......Зашёл в функцию parentheses. Текущий символ: (.......Зашёл в функцию orExept. Текущий символ: (........Зашёл в функцию term. Текущий символ: (.........Зашёл в функцию parentheses. Текущий символ: (.........Зашёл в функцию orExept. Текущий символ: 1 ....Зашёл в функцию term. Текущий символ: 1 ....Зашёл в функцию parentheses. Текущий символ: 1 ....Вышел из функции parentheses. Возвращаемое значение: 1 ....Вышел из функции term. Возвращаемое значение: 1 ....Зашёл в функцию orExept. Текущий символ: 1 ....Зашёл в функцию term. Текущий символ: 1 ....Зашёл в функцию parentheses. Текущий символ: 1 ....Вышел из функции parentheses. Возвращаемое значение: 1 ....Вышел из функции term. Возвращаемое значение: 1 ....Вышел из функции orExept. Возвращаемое значение: 1 ....Вышел из функции orExept. Возвращаемое значение: 2 ....Вышел из функции parentheses. Возвращаемое значение: 2 ....Зашёл в функцию term. Текущий символ: (.........Зашёл в функцию parentheses. Текущий символ: (..........Зашёл в функцию orExept. Текущий символ: 0 ....Зашёл в функцию term. Текущий символ: 0 ....Зашёл в функцию parentheses. Текущий символ: 0 ....Вышел из функции parentheses. Возвращаемое значение: 0

		.....Вышел из функции term. Возвращаемое значение: 0 .....Зашёл в функцию orExept. Текущий символ: 1 .....Зашёл в функцию term. Текущий символ: 1 .....Зашёл в функцию parentheses. Текущий символ: 1 .....Вышел из функции parentheses. Возвращаемое значение: 1 .....Вышел из функции term. Возвращаемое значение: 1 .....Вышел из функции orExept. Возвращаемое значение: 1 .....Вышел из функции orExept. Возвращаемое значение: 1 .....Вышел из функции parentheses. Возвращаемое значение: 1 .....Зашёл в функцию term. Текущий символ: 1 .....Зашёл в функцию parentheses. Текущий символ: 1 .....Вышел из функции parentheses. Возвращаемое значение: 1 .....Вышел из функции term. Возвращаемое значение: 1 .....Вышел из функции term. Возвращаемое значение: 1 .....Вышел из функции term. Возвращаемое значение: 2 .....Вышел из функции orExept. Возвращаемое значение: 2 .....Вышел из функции parentheses. Возвращаемое значение: 2 .....Вышел из функции term. Возвращаемое значение: 2 ....Вышел из функции orExept. Возвращаемое значение: 2 ...Вышел из функции nparentheses. Возвращаемое значение: 0 ..Вышел из функции term. Возвращаемое значение: 0 .Вышел из функции orExept. Возвращаемое значение: 0 Вышел из функции state. Возвращаемое значение: FALSE Значение выражения: FALSE

6.	FALSE	Зашёл в функцию state. Аргумент: 0 .Зашёл в функцию orExept. Текущий символ: 0 ..Зашёл в функцию term. Текущий символ: 0 ...Зашёл в функцию parentheses. Текущий символ: 0 ...Вышел из функции parentheses. Возвращаемое значение: 0 ..Вышел из функции term. Возвращаемое значение: 0 .Вышел из функции orExept. Возвращаемое значение: 0 Вышел из функции state. Возвращаемое значение: FALSE Значение выражения: FALSE
7.	NOT TRUE OR NOT TRUE	Зашёл в функцию state. Аргумент: `1|`1 .Зашёл в функцию orExept. Текущий символ: ` ..Зашёл в функцию term. Текущий символ: ` ...Зашёл в функцию nparentheses. Текущий символ: ` ...Вышел из функции nparentheses. Возвращаемое значение: 0 ..Вышел из функции term. Возвращаемое значение: 0 ..Зашёл в функцию orExept. Текущий символ: ` ...Зашёл в функцию term. Текущий символ: ` ....Зашёл в функцию nparentheses. Текущий символ: ` ....Вышел из функции nparentheses. Возвращаемое значение: 0 ...Вышел из функции term. Возвращаемое значение: 0 ..Вышел из функции orExept. Возвращаемое значение: 0 .Вышел из функции orExept. Возвращаемое значение: 0 Вышел из функции state. Возвращаемое значение: FALSE Значение выражения: FALSE

Выводы.

Написана программа реализующая рекурсивный спуск, обратились к формам Бэкуса-Наура для решения поставленной задачи. Основательно ознакомились с рекурсией и появились навыки парсинга предложений.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

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

namespace main_vars {
    unsigned int str_index;
    char *input_str = nullptr;
}
//объявляем заранее что-бы у каждой функции была видимость другой функции
string state(char* input, int n, ofstream &fout);
int orExept(symbolFieald &symb, int n, ofstream &fout);
int term(symbolFieald &symb, int n, ofstream &fout);
int parentheses(symbolFieald &symb, int n, ofstream &fout);
int nparentheses(symbolFieald &symb, int n, ofstream &fout);
char interpretator(const char* str);
symbolFieald getNext();


int main() { /* головная функция, тут реализован ввод и вывод из файла, преобразование строки из слов в
     в символьную и вызов функции state, которая является точкой запуска рекурсивного алгоритма */
    ifstream fin;
    ofstream fout;
    string path1, path2;
    cout << "Введите путь откуда читать логическое выражение и путь куда записывать:\n";
    //работа с файлами
    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] = interpretator(pch);
        count++;
        pch = strtok(nullptr, " ");
    }
    str[count] = '\0';
    string result = state(str, 0, fout);
    cout << "Значение выражения: " << result <<"\n";
    fout << "Значение выражения: " << result <<"\n";
    delete [] str;
    delete [] input_str_cpy;
    fout.close();

    return 0;
}
void printDepth(int n, ofstream& fout){ // Для вывода глубины рекурсии
    for(int i = 0; i < n; i++)
        fout << ".";
}
```

```
string state(char* input, int n, ofstream &fout) { /* Проверяем первый символ строки, после чего
 * запускается алгоритм рекурсивного спуска и вычисляется результат выражения */
    printDepth(n, fout);
    fout << "Зашёл в функцию state. " << "Аргумент: " << input << "\n";
    int result = 0;
    symbolFieald symb;
    main_vars::input_str = input;
    main_vars::str_index = 0;
    symb = getNext();
    switch (symb.attrib) {
        case LP:
        case NUM:
        case NOT:
            // orExept может начинаться с открывающейся скобки,  числа или слова not
            result = orExept(symb, n + 1, fout);
            break;
        default:
            break;
    }
    printDepth(n, fout);
    fout << "Вышел из функции state. Возвращаемое значение: ";
    if (result > 0) {
        fout << "TRUE\n";
        return "TRUE";
    } else {
        fout << "FALSE\n";
        return "FALSE";
    }
}

int orExept(symbolFieald &symb, int n, ofstream &fout) { // разбирает значения, между которыми стоит or   expr
    printDepth(n, fout);
    fout << "Зашёл в функцию orExept. " << "Текущий символ: " << symb.symbol << "\n";
    symbolFieald 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 = getNext();
            switch (tmp_symb.attrib) {
                case OR:
                    tmp_symb = getNext();
                    ret += orExept(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;
    }
    printDepth(n, fout);
    fout << "Вышел из функции orExept. Возвращаемое значение: " << ret << "\n";
    return ret;
}

int term(symbolFieald &symb, int n, ofstream &fout) { // разбирает выражения,
между которыми стоит and
    printDepth(n, fout);
    fout << "Зашёл в функцию term. " << "Текущий символ: " << symb.symbol <<
"\n";
    symbolFieald tmp_symb = symb;
    int ret = 0;
    switch (tmp_symb.attrib) {
        case LP:
        case NUM:
            ret = parentheses(tmp_symb, n + 1, fout);
            tmp_symb = getNext();
            switch (tmp_symb.attrib) {
                case AND:
                    tmp_symb = getNext();
                    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 = nparentheses(symb, n + 1, fout);
            tmp_symb = getNext();
            switch (tmp_symb.attrib) {
                case AND:
                    tmp_symb = getNext();
                    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;
    }
    printDepth(n, fout);
    fout << "Вышел из функции term. Возвращаемое значение: " << ret << "\n";
    return ret;
}
int nparentheses(symbolFieald &symb, int n, ofstream &fout) {
    // разбирает  любые элементарные значения  и выражения в скобках, если перед
ними стоит отрицание
    printDepth(n, fout);
    fout << "Зашёл в функцию nparentheses. " << "Текущий символ: " <<
symb.symbol << "\n";
    int ret = 0;
    symbolFieald tmp_symb = getNext();
    switch (tmp_symb.attrib) {
```

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

            tmp_symb = getNext();
            ret = orExept(tmp_symb, n + 1, fout);
            tmp_symb = getNext();
            // Считываем закрывающуюся скобку
            break;
        default:
            break;
    }
    printDepth(n, fout);
    fout << "Вышел из функции parentheses. Возвращаемое значение: " << ret <<
"\n";
```

```
    return ret;
}

using namespace main_vars;

symbolFieald getNext() /* последовательно даёт доступ к символам логического
выражения ,
 * присваивает им атрибуты. За каждый запуск символ сдвигается на 1 */
{
    symbolFieald 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 interpretator(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 '*';
    }
}
```