Makefile:

```
1. rpn: main.cpp
2.     g++ -ansi -Wall main.cpp -o rpn
```

main.cpp:

```
1. #include <iostream>
2. #include <string>
3. #include <set>
4. #include <map>
5. #include <stack>
6. #include <queue>
7. #include <cassert>
8. #include <cstdlib>
9.
10. // Объявление типов.
11. // Токен (лексема):
12. typedef char Token;
13. // Стек токенов:
14. typedef std::stack<Token> Stack;
15. // Последовательность токенов:
16. typedef std::queue<Token> Queue;
17. // Множество различных токенов:
18. typedef std::set<Token> Set;
19. // Таблица значений переменных:
20. typedef std::map<Token, Token> Map;
21. // Пара переменная—значение:
22. typedef std::pair<Token, Token> VarVal;
23. // Строка символов:
24. typedef std::string String;
25.
26. // Является ли токен числом?
27. inline bool isNumber(Token t) {
28.     return t == '0' || t == '1';
29. }
30.
31. // Является ли токен переменной?
32. inline bool isVariable(Token t) {
33.     return (t >= 'A' && t <= 'Z') || (t >= 'a' && t <= 'z');
34. }
35.
36. // Является ли токен операцией?
37. inline bool isOperation(Token t) {
38.     return (t == '|' || t == '&' || t == '-' || t == '>' || t == '~');
39. }
40.
41. // Является ли токен открывающей скобкой?
42. inline bool isOpeningPar(Token t) {
43.     return t == '(';
44. }
45.
46. // Является ли токен закрывающей скобкой?
47. inline bool isClosingPar(Token t) {
48.     return t == ')';
49. }
50.
51. // Вернуть величину приоритета операции
52. // (чем больше число, тем выше приоритет)
53. inline int priority(Token op) {
54.     assert (isOperation(op));
55.     int res = 0;
56.     switch (op) {
57.         case '-':
58.             // Отрицание — наивысший приоритет
```

```
            res = 5;
            break;
        case '&':
            // Конъюнкция
            res = 4;
            break;
        case '|':
            // Дизъюнкция
            res = 3;
            break;
        case '>':
            // Импликация
            res = 2;
            break;
        case '~':
            // Эквивалентность — наинизший приоритет
            res = 1;
            break;
    }
    return res;
}

// Преобразовать последовательность токенов,
// представляющих выражение в инфиксной записи,
// в последовательность токенов, представляющих
// выражение в обратной польской записи
// (алгоритм Дейкстры «Сортировочная станция»)
Queue infixToPostfix(Queue input) {
    // Выходная последовательность (очередь вывода):
    Queue output;
    // Рабочий стек:
    Stack s;
    // Текущий входной токен:
    Token t;
    // Пока есть токены во входной последовательности:
    while (!input.empty()) {
        // Получить токен из начала входной последовательности
        t = input.front();
        input.pop();
        // Если токен — число или переменная, то:
        if (isNumber(t) || isVariable(t)) {
            // Добавить его в очередь вывода
            output.push(t);
        // Если токен — операция op1, то:
        } else if (isOperation(t)) {
            // Пока на вершине стека присутствует токен-операция op2
            // и у op1 приоритет меньше либо равен приоритету op2, то:
            while (!s.empty() && isOperation(s.top())
                && priority(t) <= priority(s.top())
            ) {
                // переложить op2 из стека в выходную очередь
                output.push(s.top());
                s.pop();
            }
            // Положить op1 в стек
            s.push(t);
        // Если токен — открывающая скобка, то:
        } else if (isOpeningPar(t)) {
            // Положить его в стек
            s.push(t);
        // Если токен — закрывающая скобка, то:
        } else if (isClosingPar(t)) {
            // Пока токен на вершине стека не является открывающей скобкой:
```

```
            while (!s.empty() && !isOpeningPar(s.top())) {
                // Перекладывать токены-операции из стека
                // в выходную очередь
                assert (isOperation(s.top()));
                output.push(s.top());
                s.pop();
            }
            // Если стек закончился до того,
            // как был встречен токен-«открывающая скобка», то:
            if (s.empty()) {
                // В выражении пропущена открывающая скобка
                throw String("Пропущена открывающая скобка!");
            } else {
                // Иначе выкинуть открывающую скобку из стека
                // (но не добавлять в очередь вывода)
                s.pop();
            }
        } else {
            // В остальных случаях входная последовательность
            // содержит токен неизвестного типа
            String msg("Неизвестный символ \'");
            msg += t + String("\'!");
            throw msg;
        }
    }
    // Токенов на входе больше нет, но ещё могут остаться токены в стеке.
    // Пока стек не пустой:
    while (!s.empty()) {
        // Если токен на вершине стека — открывающая скобка, то:
        if (isOpeningPar(s.top())) {
            // В выражении присутствует незакрытая скобка
            throw String("Незакрытая скобка!");
        } else {
            // Иначе переложить токен-операцию из стека в выходную очередь
            assert (isOperation(s.top()));
            output.push(s.top());
            s.pop();
        }
    }
    // Конец алгоритма.
    // Выдать полученную последовательность
    return output;
}

// Напечатать последовательность токенов
void printSequence(Queue q) {
    while (!q.empty()) {
        std::cout << q.front();
        q.pop();
    }
    std::cout << std::endl;
}

// Является ли символ пробельным?
inline bool isSpace(char c) {
    return c <= ' ';
}

// Если символ — маленькая буква, преобразовать её в большую,
// иначе просто вернуть этот же символ
inline char toUpperCase(char c) {
    if (c >= 'a' && c <= 'z') {
        return c - 'a' + 'A';
```

```
185.     } else {
186.         return c;
187.     }
188. }
189.
190. // Преобразовать строку с выражением в последовательность токенов
191. // (лексический анализатор)
192. Queue stringToSequence(const String &s) {
193.     Queue res;
194.     for (size_t i = 0; i < s.size(); ++i) {
195.         if (!isSpace(s[i])) {
196.             res.push(toUpperCase(s[i]));
197.         }
198.     }
199.     return res;
200. }
201.
202. // Напечатать сообщение об ошибке
203. inline void printErrorMessage(const String &err) {
204.     std::cerr << "*** ОШИБКА! " << err << std::endl;
205. }
206.
207. // Ввести выражение с клавиатуры
208. inline String inputExpr() {
209.     String expr;
210.     std::cout << "Формула логики высказываний: ";
211.     std::getline(std::cin, expr);
212.     return expr;
213. }
214.
215. // Выделить из последовательности токенов переменные
216. Set getVariables(Queue s) {
217.     Set res;
218.     while (!s.empty()) {
219.         if (isVariable(s.front()) && res.count(s.front()) == 0) {
220.             res.insert(s.front());
221.         }
222.         s.pop();
223.     }
224.     return res;
225. }
226.
227. // Получить значения переменных с клавиатуры
228. Map inputVarValues(const Set &var) {
229.     Token val;
230.     Map res;
231.     for (Set::const_iterator i = var.begin(); i != var.end(); ++i) {
232.         do {
233.             std::cout << *i << " = ";
234.             std::cin >> val;
235.             if (!isNumber(val)) {
236.                 std::cerr << "Введите 0 или 1!" << std::endl;
237.             }
238.         } while (!isNumber(val));
239.         res.insert(VarVal(*i, val));
240.     }
241.     return res;
242. }
243.
244. // Заменить переменные их значениями
245. Queue substValues(Queue expr, Map &varVal) {
246.     Queue res;
247.     while (!expr.empty()) {
```

```
248.         if (isVariable(expr.front())) {
249.             res.push(varVal[expr.front()]);
250.         } else {
251.             res.push(expr.front());
252.         }
253.         expr.pop();
254.     }
255.     return res;
256. }
257.
258. // Является ли операция бинарной?
259. inline bool isBinOp(Token t) {
260.     return t == '&' || t == '|' || t == '>' || t == '~';
261. }
262.
263. // Является ли операция унарной?
264. inline bool isUnarOp(Token t) {
265.     return t == '-';
266. }
267.
268. // Получить bool-значение токена-числа (true или false)
269. inline bool logicVal(Token x) {
270.     assert (isNumber(x));
271.     return x == '1';
272. }
273.
274. // Преобразовать bool-значение в токен-число
275. inline Token boolToToken(bool x) {
276.     if (x) {
277.         return '1';
278.     } else {
279.         return '0';
280.     }
281. }
282.
283. // Вычислить результат бинарной операции
284. inline Token evalBinOp(Token a, Token op, Token b) {
285.     assert (isNumber(a) && isBinOp(op) && isNumber(b));
286.     bool res;
287.     // Получить bool-значения операндов
288.     bool left = logicVal(a);
289.     bool right = logicVal(b);
290.     switch (op) {
291.         case '&':
292.             // Конъюнкция
293.             res = left && right;
294.             break;
295.         case '|':
296.             // Дизъюнкция
297.             res = left || right;
298.             break;
299.         case '>':
300.             // Импликация
301.             res = !left || right;
302.             break;
303.         case '~':
304.             // Эквивалентность
305.             res = (!left || right) && (!right || left);
306.             break;
307.     }
308.     return boolToToken(res);
309. }
310.
```

```
311. // Вычислить результат унарной операции
312. inline Token evalUnarOp(Token op, Token a) {
313.     assert (isUnarOp(op) && isNumber(a));
314.     bool res = logicVal(a);
315.     switch (op) {
316.         case '-':
317.             // Отрицание
318.             res = !res;
319.             break;
320.     }
321.     return boolToToken(res);
322. }
323.
324. // Вычислить значение операции, модифицируя стек.
325. // Результат помещается в стек
326. void evalOpUsingStack(Token op, Stack &s) {
327.     assert (isOperation(op));
328.     // Если операция бинарная, то:
329.     if (isBinOp(op)) {
330.         // В стеке должны быть два операнда
331.         if (s.size() >= 2) {
332.             // Если это так, то извлекаем правый операнд-число
333.             Token b = s.top();
334.             if (!isNumber(b)) {
335.                 throw String("Неверное выражение!");
336.             }
337.             s.pop();
338.             // Затем извлекаем левый операнд-число
339.             Token a = s.top();
340.             if (!isNumber(a)) {
341.                 throw String("Неверное выражение!");
342.             }
343.             s.pop();
344.             // Помещаем в стек результат операции
345.             s.push(evalBinOp(a, op, b));
346.         } else {
347.             throw String("Неверное выражение!");
348.         }
349.     // Иначе операция унарная
350.     } else if (isUnarOp(op) && !s.empty()) {
351.         // Извлекаем операнд
352.         Token a = s.top();
353.         if (!isNumber(a)) {
354.             throw String("Неверное выражение!");
355.         }
356.         s.pop();
357.         // Помещаем в стек результат операции
358.         s.push(evalUnarOp(op, a));
359.     } else {
360.         throw String("Неверное выражение!");
361.     }
362. }
363.
364. // Вычислить значение выражения, записанного в обратной польской записи
365. Token evaluate(Queue expr) {
366.     // Рабочий стек
367.     Stack s;
368.     // Текущий токен
369.     Token t;
370.     // Пока входная последовательность содержит токены:
371.     while (!expr.empty()) {
372.         // Считать очередной токен
373.         t = expr.front();
```

```
374.         assert (isNumber(t) || isOperation(t));
375.         expr.pop();
376.         // Если это число, то:
377.         if (isNumber(t)) {
378.             // Поместить его в стек
379.             s.push(t);
380.         // Если это операция, то:
381.         } else if (isOperation(t)) {
382.             // Вычислить её, модифицируя стек
383.             // (результат также помещается в стек)
384.             evalOpUsingStack(t, s);
385.         }
386.     }
387.     // Результат — единственный элемент в стеке
388.     if (s.size() == 1) {
389.         // Вернуть результат
390.         return s.top();
391.     } else {
392.         throw String("Неверное выражение!");
393.     }
394. }
395.
396. // Вывести результат вычисления на экран
397. void printResult(Token r) {
398.     assert (isNumber(r));
399.     std::cout << "Значение выражения: " << r << std::endl;
400. }
401.
402. // Главная программа
403. int main() {
404.     // Ввести выражение
405.     std::string expr = inputExpr();
406.     // Преобразовать выражение в последовательность токенов
407.     Queue input = stringToSequence(expr);
408.     // Напечатать полученную последовательность токенов
409.     // printSequence(input);
410.     try {
411.         // Преобразовать последовательность токенов в ОПЗ
412.         Queue output = infixToPostfix(input);
413.         // Напечатать полученную последовательность токенов
414.         printSequence(output);
415.         // Ввести значения переменных с клавиатуры
416.         Map varVal = inputVarValues(getVariables(output));
417.         // Подставить значения переменных в выражение
418.         Queue rpn = substValues(output, varVal);
419.         // Напечатать получившееся выражение
420.         printSequence(rpn);
421.         // Вычислить полученное выражение
422.         Token res = evaluate(rpn);
423.         // Напечатать результат
424.         printResult(res);
425.     } catch (const String &err) {
426.         // Если возникла ошибка, вывести сообщение
427.         printErrorMessage(err);
428.         // И выйти из программы с неудачным кодом завершения
429.         exit(1);
430.     }
431.     // Конец программы
432.     return 0;
433. }
```