Практическая часть

Разработать программу, решающую задачи согласно своему варианту по табл. 1.1. Программа должна считывать формулу логики высказываний в указанной нормальной форме. Алгоритмы, выполняющие решение задачи, должны содержаться в отдельном модуле.

1. Программа должна строить полную таблицу истинности введённой формулы. (КНФ)

4. Программа должна отыскивать все интерпретации, на которых введённая формула принимает истинное значение. (КНФ)

```
#include <iostream>
#include <string>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <cassert>
#include <cstdlib>
#include <windows.h>
#include <iomanip>

// Объявление типов.
// Токен (лексема):
typedef char Token;
// Стек токенов:
typedef std::stack<Token> Stack;
// Последовательность токенов:
typedef std::queue<Token> Queue;
// Множество различных токенов:
typedef std::set<Token> Set;
// Таблица значений переменных:
typedef std::map<Token, Token> Map;
// Пара переменная—значение:
typedef std::pair<Token, Token> VarVal;
// Строка символов:
typedef std::string String;
using TruthTable = std::map<Queue, Token>;

// Является ли токен числом?
inline bool isNumber(Token t) {
  return t == '0' || t == '1';
}

// Является ли токен переменной?
inline bool isVariable(Token t) {
  return (t >= 'A' && t <= 'Z') || (t >= 'a' && t <= 'z');
}

// Является ли токен операцией?
inline bool isOperation(Token t) {
  return (t == '|' || t == '&' || t == '-' || t == '>' || t == '~');
}

// Является ли токен открывающей скобкой?
inline bool isOpeningPar(Token t) {
  return t == '(';
}

// Является ли токен закрывающей скобкой?
```

```
inline bool isClosingPar(Token t) {
  return t == ')';
}

// Вернуть приоритет операции
inline int priority(Token op) {
  assert  (isOperation(op));
  int res = 0;
  switch (op) {
    case '-':
      // 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)) {
      output.push(t);
      // Если токен — операция op1, то:
    } else if (isOperation(t)) {
      while (!s.empty() && isOperation(s.top())
            && priority(t) <= priority(s.top())
        ) {
        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 была встречена открывающаяся скобка:
      if (s.empty()) {
        throw String("Пропущена открывающая скобка!");
      } else {
        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()) {
    if (isOpeningPar(s.top())) {
      throw String("Незакрытая скобка!");
    } else {
      assert (isOperation(s.top()));
      output.push(s.top());
      s.pop();
    }
  }
  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';
```

```
  } else {
    return c;
  }
}

// Преобразовать строку с выражением в последовательность токенов
// (лексический анализатор)
Queue stringToSequence(const String &s) {
  Queue res;
  for (size_t i = 0; i < s.size(); ++i) {
    if (!isSpace(s[i])) {
      res.push(toUpperCase(s[i]));
    }
  }
  return res;
}

// Вывести сообщение об ошибке
inline void printErrorMessage(const String &err) {
  std::cerr << "*** ОШИБКА! " << err << std::endl;
}

// Ввод выражения с клавиатуры
inline String inputExpr() {
  String expr;
  std::cout << "Формула логики высказываний: ";
  std::getline(std::cin, expr);
  return expr;
}

// Выделить из последовательности токенов переменные
Set getVariables(Queue s) {
  Set res;
  while (!s.empty()) {
    if (isVariable(s.front()) && res.count(s.front()) == 0) {
      res.insert(s.front());
    }
    s.pop();
  }
  return res;
}

// Ввод значений переменных с клавиатуры
Map inputVarValues(const Set &var) {
  Token val;
  Map res;
  for (char i: var) {
    do {
      std::cout << i << " = ";
      std::cin >> val;
      if (!isNumber(val)) {
        std::cerr << "Введите 0 или 1!" << std::endl;
      }
    } while (!isNumber(val));
    res.insert(VarVal(i, val));
  }
  return res;
}

// Заменить переменные их значениями
Queue substValues(Queue expr, Map &varVal) {
  Queue res;
```

```
  while (!expr.empty()) {
    if (isVariable(expr.front())) {
      res.push(varVal[expr.front()]);
    } else {
      res.push(expr.front());
    }
    expr.pop();
  }
  return res;
}

// Является ли операция бинарной?
inline bool isBinOp(Token t) {
  return t == '&' || t == '|' || t == '>' || t == '~';
}

// Является ли операция унарной?
inline bool isUnarOp(Token t) {
  return t == '-';
}

// Получить bool-значение токена-числа (true или false)
inline bool logicVal(Token x) {
  assert (isNumber(x));
  return x == '1';
}

// Преобразовать bool-значение в токен-число
inline Token boolToToken(bool x) {
  if (x) {
    return '1';
  } else {
    return '0';
  }
}

// Вычислить результат бинарной операции
inline Token evalBinOp(Token a, Token op, Token b) {
  assert (isNumber(a) && isBinOp(op) && isNumber(b));
  bool res;
  // Получить bool-значения операндов
  bool left = logicVal(a);
  bool right = logicVal(b);
  switch (op) {
    case '&':
      // Конъюнкция
      res = left && right;
      break;
    case '|':
      // Дизъюнкция
      res = left || right;
      break;
    case '>':
      // Импликация
      res = !left || right;
      break;
    case '~':
      // Эквивалентность
      res = (!left || right) && (!right || left);
      break;
  }
  return boolToToken(res);
```

```
}

// Вычислить результат унарной операции
inline Token evalUnarOp(Token op, Token a) {
  assert  (isUnarOp(op) && isNumber(a));
  bool res = logicVal(a);
  switch (op) {
    case '-':
      // Отрицание
      res = !res;
      break;
  }
  return boolToToken(res);
}

// Вычислить значение операции, модифицируя стек.
// Результат помещается в стек
void evalOpUsingStack(Token op, Stack &s) {
  assert  (isOperation(op));
  // Если операция бинарная, то:
  if (isBinOp(op)) {
    // В стеке должны быть два операнда
    if (s.size() >= 2) {
      // Если это так, то извлекаем правый операнд-число
      Token b = s.top();
      if (!isNumber(b)) {
        throw String("Неверное выражение!");
      }
      s.pop();
      // Затем извлекаем левый операнд-число
      Token a = s.top();
      if (!isNumber(a)) {
        throw String("Неверное выражение!");
      }
      s.pop();
      // Помещаем в стек результат операции
      s.push(evalBinOp(a, op, b));
    } else {
      throw String("Неверное выражение!");
    }
    // Иначе операция унарная
  } else if (isUnarOp(op) && !s.empty()) {
    // Извлекаем операнд
    Token a = s.top();
    if (!isNumber(a)) {
      throw String("Неверное выражение!");
    }
    s.pop();
    // Помещаем в стек результат операции
    s.push(evalUnarOp(op, a));
  } else {
    throw String("Неверное выражение!");
  }
}

// Вычислить значение выражения, записанного в обратной польской записи
Token evaluate(Queue expr) {
  // Рабочий стек
  Stack s;
  // Текущий токен
  Token t;
  // Пока входная последовательность содержит токены:
```

```
  while (!expr.empty()) {
    // Считать очередной токен
    t = expr.front();
    assert  (isNumber(t) || isOperation(t));
    expr.pop();
    // Если это число, то:
    if (isNumber(t)) {
      // Поместить его в стек
      s.push(t);
      // Если это операция, то:
    } else if (isOperation(t)) {
      // Вычислить её, модифицируя стек
      // (результат также помещается в стек)
      evalOpUsingStack(t, s);
    }
  }
  // Результат — единственный элемент в стеке
  if (s.size() == 1) {
    // Вернуть результат
    return s.top();
  } else {
    throw String("Неверное выражение!");
  }
}

// Вывести результат вычисления на экран
void printResult(Token r) {
  assert  (isNumber(r));
  std::cout << "Значение выражения: " << r << std::endl;
}

// Возвращает результат возведения в степень power числа number
// Алгоритм быстрого возведения в натуральную степень
template<typename T>
T pow(T number, size_t power) {
  T res = 1;
  T currPowOf2 = number;
  while (power) {
    if (power & 1)
      res *= currPowOf2;
    currPowOf2 *= currPowOf2;
    power >>= 1;
  }
  return res;
}

// Возвращает очередь с элементами множества set
Queue setToQueue(const Set &set) {
  Queue res;
  for (char it: set)
    res.push(it);
  return res;
}

// Возвращает таблицу истинности для заданных
// переменных vars и выражения output в постфиксной форме
TruthTable getTruthTable(const Queue &output,
                         const Set &vars) {
  auto varsN = vars.size();
  auto maxNum = pow(2, varsN);
  Map substitutes;
  TruthTable res;
```

```
  auto expr = setToQueue(vars);
  for (int i = 0; i < maxNum; ++i) {
    auto var = vars.rbegin();
    int bit = i;
    while (var != vars.rend()) {
      substitutes[*var] = boolToToken(bit & 1);
      var++;
      bit >>= 1;
    }
    Queue rpn = substValues(output, substitutes);
    Queue rpn_vars = substValues(expr, substitutes);
    res[rpn_vars] = evaluate(rpn);
  }
  return res;
}

// Выводит на экран таблицу истинности table
// с заданными переменными vars и выводит результат выражения
// при заданном флаге expressionValueToPrint
void printTruthTable(const TruthTable &table,
                     const Set &vars = {},
                     const bool expressionValueToPrint = true) {
  if (!vars.empty()) {
    for (const auto &var: vars)
      std::cout << var << std::setw(4);
    if (expressionValueToPrint)
      std::cout << std::setw(7) << std::right << "Result";
    std::cout << '\n';
  }
  for (const auto &row: table) {
    auto vals = row.first;
    while (!vals.empty()) {
      std::cout << vals.front() << std::setw(4);
      vals.pop();
    }
    if (expressionValueToPrint)
      std::cout << row.second;
    std::cout << '\n';
  }
}

// Возвращает новую таблицу истинности, в которой отсутствуют
// строки из table, удовлетворяющие унарному предикату p
template<typename UnitPredicate>
TruthTable deleteRowsIf(const TruthTable &table,
                        UnitPredicate p) {
  TruthTable res;
  for (const auto &pair: table) {
    if (!p(pair))
      res.insert(pair);
  }
  return res;
}

int main() {
  SetConsoleOutputCP(CP_UTF8);
  // Ввод
  std::string expr = inputExpr();
  Queue input = stringToSequence(expr);
  try {
    Queue output = infixToPostfix(input);
    printSequence(output);
```

```
    auto vars = getVariables(output);
    // Задание 1
    auto table = getTruthTable(output, vars);
    printTruthTable(table, vars);
    // Задание 4
    auto newTable = deleteRowsIf(table,
                                 [](const auto &pair) {
                                   return pair.second == '0';
                                 });
    std::cout << '\n';
    printTruthTable(newTable, vars, false);
  } catch (const String &err) {
    printErrorMessage(err);
    exit(1);
  }
  return 0;
}
```

Результат работы программы:

1.

```
Формула логики высказываний:X&(-Y|Z)
 XY-Z|&
X   Y   Z Result
0   0   0   0
0   0   1   0
0   1   0   0
0   1   1   0
1   0   0   1
1   0   1   1
1   1   0   0
1   1   1   1
```

4.

```
X   Y   Z
1   0   0
1   0   1
1   1   1

Process finished with exit code 0
```

Вывод: в ходе работы была изучена логика высказываний и закреплены навыки решения теоретических и практических задач.