Написать решения двух задач по математической логике на языке Java. Имеется решение задач на языке C++. Можно переработать данные решения, но так, чтобы это было максимально незаметно. Решение (код) прокомментировать. Вариант решения и условия задач в файле.

1 задача

|  |
| --- |
| #include <iostream> |
|  | #include <fstream> |
|  |  |
|  | using namespace std; |
|  |  |
|  | string expressionParser(string); |
|  |  |
|  | string negationParser(string s) { |
|  | if (s[0] == '(') |
|  | return expressionParser(s.substr(1, s.length() - 2)); |
|  | if (s[0] == '!') return "(!" + negationParser(s.substr(1)) + ")"; else return s; |
|  | } |
|  |  |
|  | string conjunctionParser(string s) { |
|  | int brackets\_count = 0; |
|  | for (int i = s.length() - 1; i >= 0; i--) |
|  | if (s[i] == '(') brackets\_count++; |
|  | else if (s[i] == ')') brackets\_count--; |
|  | else if (brackets\_count == 0 && s[i] == '&') { |
|  | return "(&," + conjunctionParser(s.substr(0, i)) + "," + negationParser(s.substr(i + 1)) + ")"; |
|  | } |
|  | return negationParser(s); |
|  | } |
|  |  |
|  | string disjunctionParser(string s) { |
|  | int brackets\_count = 0; |
|  | for (int i = s.length() - 1; i > 0; i--) |
|  | if (s[i] == '(') brackets\_count++; |
|  | else if (s[i] == ')') brackets\_count--; |
|  | else if (brackets\_count == 0 && s[i] == '|') |
|  | return "(|," + disjunctionParser(s.substr(0, i)) + "," + conjunctionParser(s.substr(i + 1)) + ")"; |
|  | return conjunctionParser(s); |
|  | } |
|  |  |
|  | string expressionParser(string s) { |
|  | int brackets\_count = 0; |
|  | for (int i = 0; i < s.length() - 1; i++) { |
|  | if (s[i] == '(') brackets\_count++; |
|  | else if (s[i] == ')') brackets\_count--; |
|  | else if (brackets\_count == 0 && s[i] == '-' && s[i + 1] == '>') |
|  | return "(->," + disjunctionParser(s.substr(0, i)) + "," + expressionParser(s.substr(i + 2)) + ")"; |
|  | } |
|  | return disjunctionParser(s); |
|  | } |
|  |  |
|  | int main() { |
|  | std::ios::sync\_with\_stdio(false); |
|  | //ifstream cin("input.txt"); |
|  | //ofstream cout("output.txt"); |
|  | string s; |
|  | getline(cin, s); |
|  | long long j = 0; |
|  | while (j < s.length()) { |
|  | switch(s[j]) { |
|  | case ')': |
|  | case '(': |
|  | case '!': |
|  | case '>': |
|  | case '-': |
|  | case '|': |
|  | case '&': |
|  | case '0': |
|  | case '1': |
|  | case '2': |
|  | case '3': |
|  | case '4': |
|  | case '5': |
|  | case '6': |
|  | case '7': |
|  | case '8': |
|  | case '9': |
|  | break; |
|  | default: { |
|  | char c = s[j]; |
|  | int checkcode = (int) c; |
|  | if (!(checkcode >= 65 && checkcode <= 90)) s.erase(j, 1); |
|  | } |
|  | } |
|  | j++; |
|  | } |
|  | string ans = expressionParser(s); |
|  | cout << ans << endl; |
|  | return 0; |
|  | } |

2 задача

|  |
| --- |
| #include <iostream> |
|  | #include <fstream> |
|  | #include <vector> |
|  | #include <map> |
|  |  |
|  | using namespace std; |
|  |  |
|  | string axioms[10] = {"(->,A,(->,B,A))", "(->,(->,A,B),(->,(->,A,(->,B,C)),(->,A,C)))", "(->,A,(->,B,(&,A,B)))", |
|  | "(->,(&,A,B),A)", "(->,(&,A,B),B)", "(->,A,(|,A,B))", "(->,B,(|,A,B))", |
|  | "(->,(->,A,C),(->,(->,B,C),(->,(|,A,B),C)))", "(->,(->,A,B),(->,(->,A,(!B)),(!A)))", |
|  | "(->,(!(!A)),A)"}; |
|  |  |
|  | vector<string> lines, expressionlines; |
|  |  |
|  | map<string, int> assumptions; |
|  |  |
|  | struct node { |
|  | string value; |
|  | node \*left; |
|  | node \*right; |
|  | }; |
|  |  |
|  | vector<node \*> axioms\_trees(10), lines\_trees; |
|  |  |
|  | string checkline(string s) { |
|  | long long j = 0; |
|  | while (j < s.length()) { |
|  | if (s[j] == ' ' || s[j] == '\t') { |
|  | s.erase(j, 1); |
|  | j--; |
|  | } |
|  | j++; |
|  | } |
|  | return s; |
|  | } |
|  |  |
|  | string expressionParser(string); |
|  |  |
|  | string negationParser(string s) { |
|  | if (s.length() == 0) return ""; |
|  | if (s[0] == '(') |
|  | return expressionParser(s.substr(1, s.length() - 2)); |
|  | if (s[0] == '!') return "(!" + negationParser(s.substr(1)) + ")"; else return s; |
|  | } |
|  |  |
|  | string conjunctionParser(string s) { |
|  | int brackets\_count = 0; |
|  | if (s.length() == 0) return ""; |
|  | for (int i = s.length() - 1; i >= 0; i--) |
|  | if (s[i] == '(') brackets\_count++; |
|  | else if (s[i] == ')') brackets\_count--; |
|  | else if (brackets\_count == 0 && s[i] == '&') { |
|  | return "(&," + conjunctionParser(s.substr(0, i)) + "," + negationParser(s.substr(i + 1)) + ")"; |
|  | } |
|  | return negationParser(s); |
|  | } |
|  |  |
|  | string disjunctionParser(string s) { |
|  | int brackets\_count = 0; |
|  | if (s.length() == 0) return ""; |
|  | for (int i = s.length() - 1; i >= 0; i--) |
|  | if (s[i] == '(') brackets\_count++; |
|  | else if (s[i] == ')') brackets\_count--; |
|  | else if (brackets\_count == 0 && s[i] == '|') |
|  | return "(|," + disjunctionParser(s.substr(0, i)) + "," + conjunctionParser(s.substr(i + 1)) + ")"; |
|  | return conjunctionParser(s); |
|  | } |
|  |  |
|  | string expressionParser(string s) { |
|  | int brackets\_count = 0; |
|  | if (s.length() == 0) return ""; |
|  | for (int i = 0; i < s.length(); i++) { |
|  | if (s[i] == '(') brackets\_count++; |
|  | else if (s[i] == ')') brackets\_count--; |
|  | else if (brackets\_count == 0 && s[i] == '-' && i < s.length() - 1 && s[i + 1] == '>') |
|  | return "(->," + disjunctionParser(s.substr(0, i)) + "," + expressionParser(s.substr(i + 2)) + ")"; |
|  | } |
|  | return disjunctionParser(s); |
|  | } |
|  |  |
|  | void add(string s, node \*tree) { |
|  | if (s[0] == '(') { |
|  | s.erase(0, 1); |
|  | s.erase(s.length() - 1, 1); |
|  | } |
|  | int start = 0; |
|  | int brackets\_count = 0; |
|  | string left\_s = "", right\_s = "", val = ""; |
|  | bool part = false; |
|  | if (s[0] == '-') { |
|  | start = 3; |
|  | val = "->"; |
|  | } else { |
|  | start = 2; |
|  | val = s[0]; |
|  | } |
|  | for (int i = start; i < s.length(); i++) { |
|  | if (s[i] == '(') brackets\_count++; |
|  | else if (s[i] == ')') brackets\_count--; |
|  | if (brackets\_count == 0 && s[i] == ',') part = true; |
|  | else if (!part) left\_s += s[i]; else right\_s += s[i]; |
|  | } |
|  | tree->left = new node; |
|  | tree->right = new node; |
|  | if (val == "&" || val == "|" || val == "->") { |
|  | tree->value = val; |
|  | add(left\_s, tree->left); |
|  | add(right\_s, tree->right); |
|  | } else if (val == "!") { |
|  | tree->value = val; |
|  | add(s.substr(1, s.length() - 1), tree->left); |
|  | tree->right = nullptr; |
|  | } else { |
|  | tree->value = val; |
|  | tree->left = nullptr; |
|  | tree->right = nullptr; |
|  | } |
|  | } |
|  |  |
|  | void create\_expression\_tree() { |
|  | for (int i = 0; i < lines.size(); i++) { |
|  | node \*line = new node; |
|  | expressionlines.push\_back(expressionParser(lines[i])); |
|  | add(expressionlines[i], line); |
|  | lines\_trees.push\_back(line); |
|  | } |
|  | } |
|  |  |
|  | void create\_axioms\_tree() { |
|  | for (int i = 0; i < 10; i++) { |
|  | node \*axioma = new node; |
|  | add(axioms[i], axioma); |
|  | axioms\_trees[i] = axioma; |
|  | } |
|  | } |
|  |  |
|  | node \*a; |
|  | node \*b; |
|  | node \*c; |
|  |  |
|  | bool compare\_trees(node \*tree1, node \*tree2) { |
|  | if (tree1 == nullptr && tree2 == nullptr) return true; |
|  | if ((tree1 == nullptr && tree2 != nullptr) || (tree1 != nullptr && tree2 == nullptr)) return false; |
|  | if (tree1->value == tree2->value) |
|  | return compare\_trees(tree1->left, tree2->left) & compare\_trees(tree1->right, tree2->right); |
|  | else return false; |
|  | } |
|  |  |
|  | bool axiom\_search(node \*tree, node \*axiom\_tree) { |
|  | if (axiom\_tree == nullptr && tree == nullptr) return true; |
|  | if ((axiom\_tree == nullptr && tree != nullptr) || (axiom\_tree != nullptr && tree == nullptr)) return false; |
|  | if (axiom\_tree->value == "A") { |
|  | if (a == nullptr) { |
|  | a = new node; |
|  | a->value = tree->value; |
|  | a->left = tree->left; |
|  | a->right = tree->right; |
|  | return true; |
|  | } else { |
|  | return compare\_trees(a, tree); |
|  | } |
|  | } else if (axiom\_tree->value == "B") { |
|  | if (b == nullptr) { |
|  | b = new node; |
|  | b->value = tree->value; |
|  | b->left = tree->left; |
|  | b->right = tree->right; |
|  | return true; |
|  | } else { |
|  | return compare\_trees(b, tree); |
|  | } |
|  | } else if (axiom\_tree->value == "C") { |
|  | if (c == nullptr) { |
|  | c = new node; |
|  | c->value = tree->value; |
|  | c->left = tree->left; |
|  | c->right = tree->right; |
|  | return true; |
|  | } else { |
|  | return compare\_trees(c, tree); |
|  | } |
|  | } else if ((axiom\_tree->value == "&" || axiom\_tree->value == "|" || axiom\_tree->value == "->") && |
|  | (axiom\_tree->value == tree->value)) { |
|  | return axiom\_search(tree->left, axiom\_tree->left) & axiom\_search(tree->right, axiom\_tree->right); |
|  | } else if (axiom\_tree->value == tree->value) { |
|  | return axiom\_search(tree->left, axiom\_tree->left); |
|  | } else return false; |
|  | } |
|  |  |
|  | int axioms\_check(int cur) { |
|  | node \*tmp = lines\_trees[cur]; |
|  | for (int i = 0; i < 10; i++) { |
|  | node \*axiom\_tmp = axioms\_trees[i]; |
|  | a = nullptr; |
|  | b = nullptr; |
|  | c = nullptr; |
|  | if (axiom\_search(tmp, axiom\_tmp)) { |
|  | free(a); |
|  | free(b); |
|  | free(c); |
|  | return i + 1; |
|  | } |
|  | free(a); |
|  | free(b); |
|  | free(c); |
|  | } |
|  | return -1; |
|  | } |
|  |  |
|  | map<string, vector<int>> right\_part; |
|  | map<string, int> all\_part; |
|  | map<int, string> left\_part; |
|  |  |
|  | int assump\_count = 1; |
|  |  |
|  | void mp\_add(string s, int ind) { |
|  | if (s[0] == '(') { |
|  | s.erase(0, 1); |
|  | s.erase(s.length() - 1, 1); |
|  | } |
|  | if (s[0] == '-') { |
|  | int start = 3; |
|  | int brackets\_count = 0; |
|  | string left\_s = "", right\_s = ""; |
|  | bool part = false; |
|  | for (int i = start; i < s.length(); i++) { |
|  | if (s[i] == '(') brackets\_count++; |
|  | else if (s[i] == ')') brackets\_count--; |
|  | if (brackets\_count == 0 && s[i] == ',') part = true; |
|  | else if (!part) left\_s += s[i]; else right\_s += s[i]; |
|  | } |
|  | right\_part[right\_s].push\_back(ind); |
|  | left\_part[ind] = left\_s; |
|  | } |
|  | } |
|  |  |
|  | void get\_annotation(int cur) { |
|  | string ss = expressionlines[cur]; |
|  | if (assumptions.count(ss) > 0) { |
|  | cout << " (Предп. " << assump\_count << ")\n"; |
|  | assump\_count++; |
|  | all\_part[ss] = cur; |
|  | mp\_add(ss, cur); |
|  | return; |
|  | } |
|  | int axioms\_ans = axioms\_check(cur); |
|  | if (axioms\_ans != -1) { |
|  | cout << " (Сх. акс. " << axioms\_ans << ")\n"; |
|  | all\_part[ss] = cur; |
|  | mp\_add(ss, cur); |
|  | return; |
|  | } |
|  | if (right\_part.count(ss) > 0) { |
|  | for (int i = 0; i < right\_part[ss].size(); i++) { |
|  | int first = right\_part[ss][i]; |
|  | if (left\_part.count(first) > 0) { |
|  | string left\_s = left\_part[first]; |
|  | if (all\_part.count(left\_s) > 0) { |
|  | int second = all\_part[left\_s]; |
|  | cout << " (M.P. " << first + 1 << ", " << second + 1 << ")\n"; |
|  | all\_part[ss] = cur; |
|  | mp\_add(ss, cur); |
|  | return; |
|  | } |
|  | } |
|  | } |
|  | } |
|  | all\_part[ss] = cur; |
|  | mp\_add(ss, cur); |
|  | cout << " (Не доказано)\n"; |
|  | } |
|  |  |
|  | int main() { |
|  | std::ios::sync\_with\_stdio(false); |
|  | freopen("input.txt", "r", stdin); |
|  | freopen("output.txt", "w", stdout); |
|  | string start\_s; |
|  | create\_axioms\_tree(); |
|  | getline(cin, start\_s); |
|  | long long j = 0; |
|  | start\_s = checkline(start\_s); |
|  | string s; |
|  | while (true) { |
|  | if (start\_s[j] == ',') { |
|  | string assumption = start\_s.substr(0, j); |
|  | start\_s.erase(0, j + 1); |
|  | if (assumption.length() > 0) assumptions[expressionParser(assumption)] = assumptions.size(); |
|  | j = -1; |
|  | } else if (start\_s[j] == '|' && start\_s[j + 1] == '-') { |
|  | string assumption = start\_s.substr(0, j); |
|  | if (assumption.length() > 0) assumptions[expressionParser(assumption)] = assumptions.size(); |
|  | break; |
|  | } |
|  | j++; |
|  | } |
|  | while (getline(cin, s)) { |
|  | s = checkline(s); |
|  | if (s.length() > 0) lines.push\_back(s); |
|  | } |
|  | create\_expression\_tree(); |
|  | for (int i = 0; i < lines.size(); i++) { |
|  | cout << "(" << i + 1 << ") " << lines[i]; |
|  | get\_annotation(i); |
|  | } |
|  | return 0; |
|  | } |

Изображение выглядит как текст, монитор, снимок экрана, компьютер

Автоматически созданное описание











