Compiling lab2.cpp:
Source file timestamp: 2016-02-23 11:36:50
Compiled at: 2016-02-23 11:47:09

```
 1. /**
 2. ==============   TOLSA   ===================
 3. ===============  LAB 2  ====================
 4. ============= Variant 16 ===================
 5. ====== Author: Oleg Pedorenko, IP-31 =======
 6. ============= FICT, ASOIU ==================
 7. ========== Created on: 23.02.2016 ==========
 8. ============================================
 9. */
10.
11. #include "iostream"
12. #include "fstream"
13. #include "string"
14. #include "vector"
15. #include "stack"
16.
17. using namespace std;
18.
19. void printHelp(string exeName)
20. {
21.    cout << "==============   TOLSA   ====================\n" <<
22.            "===============  LAB 2  ====================\n" <<
23.            "============= Variant 16 ===================\n" <<
24.            "====== Author: Oleg Pedorenko, IP-31 =======\n" <<
25.            "============= FICT, ASOIU ==================\n" <<
26.            "========== Created on: 23.02.2016 ==========\n" <<
27.            "============================================\n";
28.    cout << endl;
29.    cout << "Pass the path to the file with a list of lines to\n" <<
30.            "analyze as a command line argument\n";
31.    cout << "Pass the name of the file where you want to direct the output as\n" <<
32.            "the second argument";
33.    cout << endl;
34.    cout << "Example:\n" << exeName << " sample.pas\n";
35.    cout << endl;
36.    cout << "Output: \nAAAABBBBCC\nThis word exists in given language\n";
37.    cout << endl;
38. }
39.
40. vector<string>* loadFile(string fileName)
41. {
42.    ifstream inputFile(fileName.c_str());
43.    if(!inputFile.good())
44.    {
45.        return NULL;
46.    }
47.    vector<string>* stringVector = new vector<string>();
48.    do
49.    {
50.        stringVector->push_back("");
51.        getline(inputFile, stringVector->back());
52.    } while (inputFile.good());
53.    return stringVector;
54. }
55.
56. void printStringVector(vector<string>* stringVector, ostream& os = cout)
57. {
```

```
 58.    for(string& s: *stringVector)
 59.    {
 60.        os << s << endl;
 61.    }
 62. }
 63.
 64. int orderValue(char a)
 65. {
 66.    switch(a)
 67.    {
 68.        case '=':
 69.            return 0;
 70.            break;
 71.        case '+':
 72.        case '-':
 73.            return 1;
 74.            break;
 75.        case '*':
 76.        case '/':
 77.            return 2;
 78.            break;
 79.        default:
 80.            return -1;
 81.    }
 82. }
 83.
 84. bool precedence(char a, char b)
 85. {
 86.    if(a == '(')
 87.    {
 88.        return false;
 89.    }
 90.    if((b == '(') && (a != ')'))
 91.    {
 92.        return false;
 93.    }
 94.    if((a != '(') && (b == ')'))
 95.    {
 96.        return true;
 97.    }
 98.    if((a == '(') && (b == ')'))
 99.    {
100.        return false;
101.    }
102.    return orderValue(a) >= orderValue(b);
103. }
104.
105. bool isOperation(char a)
106. {
107.    return (orderValue(a) >= 0) || (a == '(') || (a == ')');
108. }
109.
110. string infixToPostfix(string s)
111. {
112.    string postfix = "";
113.    stack<char> opstk;
114.    for(char c: s)
115.    {
116.        if(c == ' ')
117.        {
118.            continue;
119.        }
```

```
120.        if(isOperation(c))
121.        {
122.            char smbtop;
123.            while(!opstk.empty() && precedence(smbtop = opstk.top(), c))
124.            {
125.                postfix += smbtop;
126.                opstk.pop();
127.            }
128.            if((opstk.empty()) || (c != ')'))
129.            {
130.                opstk.push(c);
131.            }
132.            else
133.            {
134.                smbtop = opstk.top();
135.                opstk.pop();
136.            }
137.        }
138.        else
139.        {
140.            postfix += c;
141.        }
142.    }
143.    while(!opstk.empty())
144.    {
145.        postfix += opstk.top();
146.        opstk.pop();
147.    }
148.    return postfix;
149. }
150.
151. vector<string>* stringVectorInfixToPostfix(vector<string>* stringVector)
152. {
153.    vector<string>* result = new vector<string>();
154.    for(string& s: *stringVector)
155.    {
156.        result->push_back(infixToPostfix(s));
157.    }
158.    return result;
159. }
160.
161. int main(int argc, char** argv)
162. {
163.    if(argc == 1)
164.    {
165.        printHelp(argv[0]);
166.    }
167.    else if(argc == 2)
168.    {
169.        vector<string>* lines = loadFile(argv[1]);
170.        if(lines == NULL)
171.        {
172.            cout << "i/o error\n";
173.        }
174.        else
175.        {
176.            printStringVector(stringVectorInfixToPostfix(lines));
177.        }
178.    }
179.    else if (argc == 3)
180.    {
181.        vector<string>* lines = loadFile(argv[1]);
```

```
    182.        ofstream os(argv[2]);
    183.        if(lines == NULL)
    184.        {
    185.            cout << "i/o error\n";
    186.        }
    187.        else
    188.        {
    189.            printStringVector(stringVectorInfixToPostfix(lines), os);
    190.        }
    191.    }
    192.    else if(argc > 3)
    193.    {
    194.        cout << "Too many arguments";
    195.    }
    196. }
196 lines: No errors
```