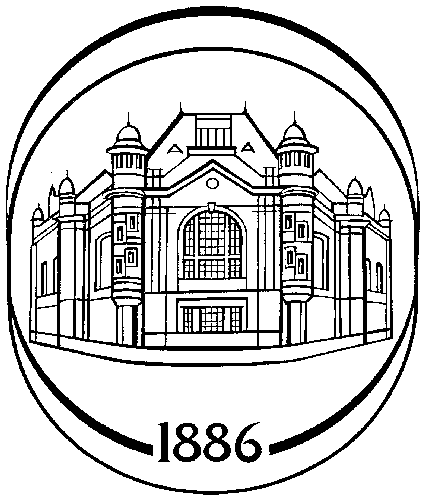
МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**Санкт-Петербургский государственный электротехнический университет “ЛЭТИ” им. В. И. Ульянова (Ленина) (СПбГЭТУ «ЛЭТИ»)**



Кафедра САПР

КУРСОВая работа

**по дисциплине «Алгоритмы и структуры данных»**

Тема: «**Преобразование алгебраических формул из инфиксной в постфиксную форму записи и вычисление значения выражения.**»

Вариант № 23 (1)

|  |  |  |
| --- | --- | --- |
| Студентка гр. 9301, ФКТИ |  | Синицкая В. А. |
| Преподаватель |  | Тутуева А. В. |

Санкт-Петербург

2020 г.

Постановка задачи

Задача для 1 варианта:

Преобразование алгебраических формул из инфиксной в постфиксную форму записи и вычисление значения выражения.

Необходимо реализовать простейшую версию калькулятора. Пользователю должен быть доступен ввод математического выражения, состоящего из чисел и арифметических знаков. Программа должна выполнить проверку корректности введенного выражения. В случае некорректного ввода необходимо вывести сообщение об ошибке с указанием позиции некорректного ввода. В противном выводится обратная польская нотация введенного выражения, а также отображается результат вычисления.

*Входные данные:*

* арифметическое выражение
* поддерживаемый тип данных: вещественные числа (double)
* поддерживаемые знаки: +, -, \*, /, ^, унарный “-”, функции с одним аргументом (cos, sin, tg, ctg, ln, log, sqrt и др. (хотя бы одну не из списка)), константы pi, e открывающая и закрывающая скобки

*Выходные данные:*

* постфиксная ФЗ
* результат вычисления

Входные данные по желанию можно читать из файла.

Структуры данных

При выполнении курсовой работы были использованы структуры данных: Element, Queue, Stack. В структуре Element есть поля для записи чисел с плавающей точкой и для строки. Queue и Stack были выбраны так как понятны и просты в использовании, а также их функционала хватает для решения поставленной задачи без повышения сложности или времени выполнения в сравнении с другими возможными вариантами. Дополнительной функцией был выбран модуль (abs), был выбран файловый ввод-вывод. Element имеет поля для записи числа типа double и для записи string, так что в одном элементе хранится операнд или оператор вместе с указанием своего приоритета. В программе написаны три основные функции: Queue\* string\_processing(ifstream&, ofstream&) – обрабатывает ввод, переводя и деля символы на элементы и проверяя ввод на верность; Queue\* convert\_to\_reverse\_polish\_notation(Queue\*, ofstream&) – принимает очередь, полученную функцией string\_processing, меняет порядок элементов, не запоминая скобки, и получает новую очередь, которая олицетворяет обратную польскую нотацию исходного выражения; double calculation(ifstream&, ofstream&) – по правилам вычисления выражения в постфиксной форме записи вычисляет и возвращает результат вычислений.

Алгоритм решения

Для решения задачи было решено использовать файловый ввод-вывод. Ввод считывается посимвольно. Для каждого символа рассмотрено, что не может быть дальше. Если встречается то, чего быть не может, например комбинация «+-», тогда в файл вывода выводится часть ввода, которую программа обработала и далее выведется сообщение об ошибке с ее кратким описанием. Унарный минус обрабатывается добавлением нуля перед минусом. В итоге ввод превращается в очередь элементов, где каждый элемент – это число, оператор или функция от одного аргумента. Если элемент – это число, то в поле для числа (поле data) лежит значение, а в поле для строки – маркер ”#”, сигнализирующий о том, что у данного элемента нет написания (поле spelling). Если элемент – операция или функция, то в поле data лежит приоритет операции (функции считаются выше степени по приоритету), а в поле spelling – написание, знак операции или слово, обозначающее функцию. Далее последовательно извлекая из полученной очереди элементы создадим новую очередь, в которой будет записана обратная польская нотация исходного выражения. По правилам перевода, используя стек для операций и функций и поля с приоритетами элементов, получим постфиксную форму записи (ФЗ). Далее, последовательно извлекая элементы из этой очереди, по правилам вычисления выражения в постфиксной ФЗ, используя стек для операндов, вычисляется значение исходного выражения.

Описание реализованных unit-тестов

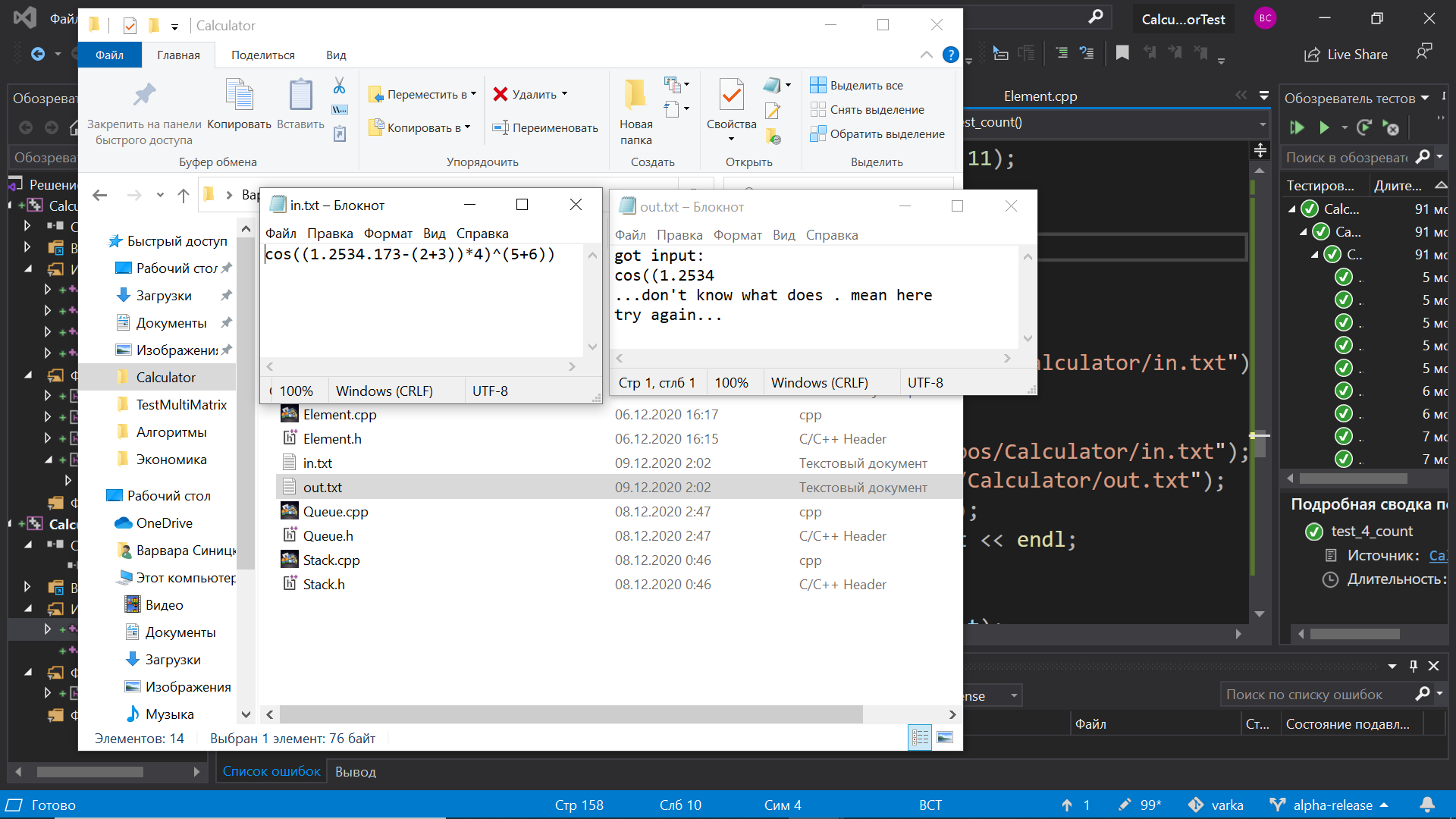
Всего был реализован 13 unit-test-ов:

Проверялась сначала корректность работы функции string\_processing, затем были рассмотрены случаи неправильного ввода, проверялось, что он выявляется и программа дальше не работает. Затем проверялась функция перевода в постфиксную ФЗ, затем уже функция вычисления результата. Тесты можно с легкостью размножить, заменяя строку ввода и верный результат, с которым нужно сравнить.

Примеры работы

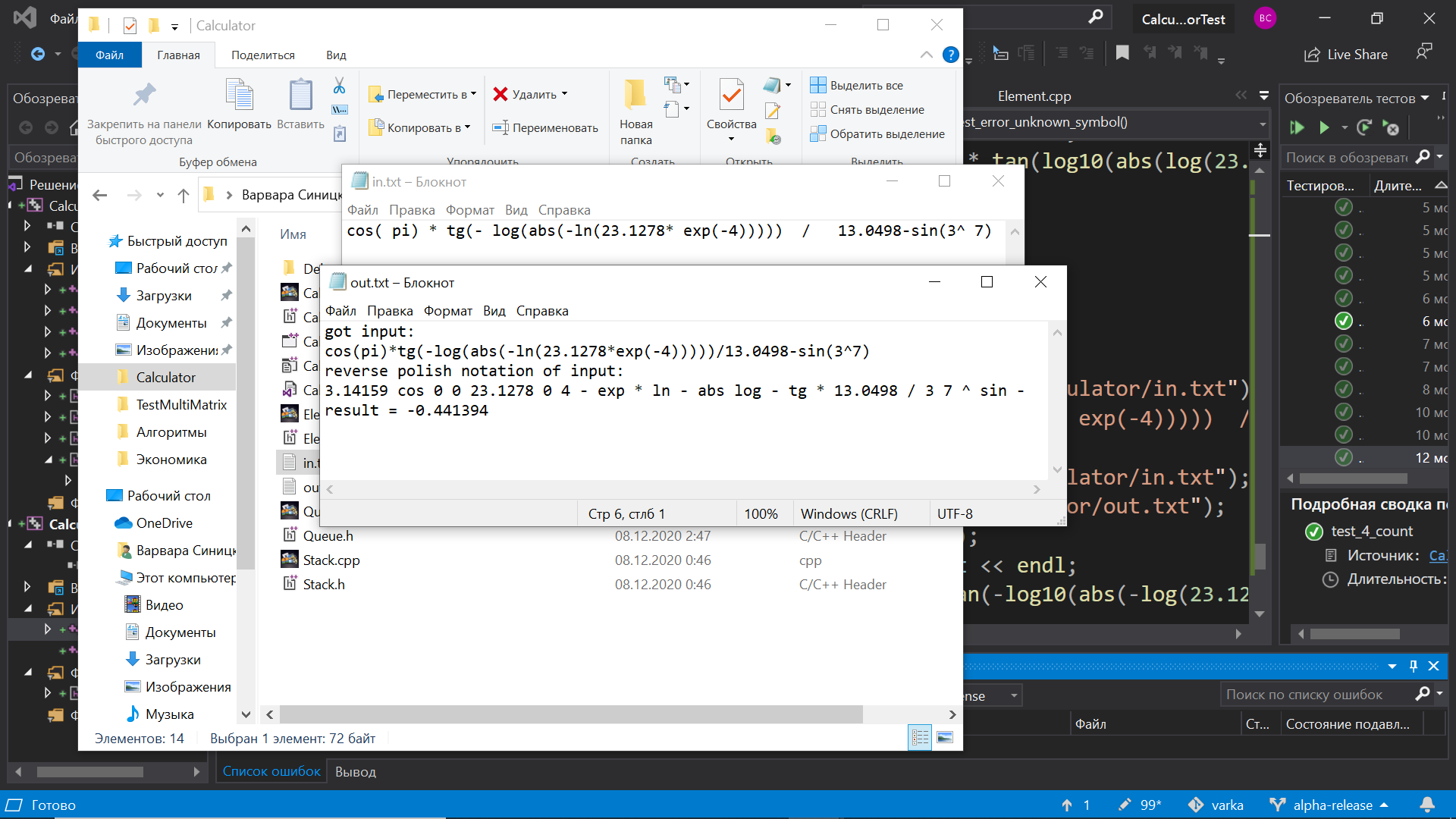
При функции main():

1. int main()
2. {
3. my\_error = false;
4. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
5. out << "cos((1.2534.173-(2+3))\*4)^(5+6))";
6. out.close();
7. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
8. out.open("/Users/varka/source/repos/Calculator/out.txt");
9. double result = calculation(in, out);
10. out.close();
11. return 0;
12. }



При функции main():

1. int main()
2. {
3. my\_error = false;
4. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
5. out << "cos( pi) \* tg(- log(abs(-ln(23.1278\* exp(-4))))) / 13.0498-sin(3^ 7) ";
6. out.close();
7. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
8. out.open("/Users/varka/source/repos/Calculator/out.txt");
9. double result = calculation(in, out);
10. out << endl << "result = " << result << endl;
11. double right\_result = cos(M\_PI) \* tan(-log10(abs(-log(23.1278 \* exp(-4))))) / 13.0498 - sin(pow(3, 7));
12. out.close();
13. return 0;
14. }



Листинг

Element.h:

1. #pragma once
2. #include <stdexcept>
3. using namespace std;
4. class Element
5. {
6. Element\* next;
7. double data;
8. string spelling;
9. void set\_next(Element\*);
10. void set\_data(double);
11. void set\_data(string);
12. void set\_data(double, string);
13. public:
14. Element();
15. Element\* get\_next();
16. double get\_data();
17. string get\_spelling();
18. ~Element();
19. friend class Queue;
20. friend class Stack;
21. //friend void string\_processing(ifstream&, int);
22. };

Element.cpp:

1. #include "Element.h"
2. Element::Element()
3. {
4. data = -1;
5. spelling = "#";
6. next = nullptr;
7. }
8. Element\* Element::get\_next() { return next; }
9. double Element::get\_data() { return data; }
10. string Element::get\_spelling() { return spelling; }
11. void Element::set\_next(Element\* new\_element) { next = new\_element; }
12. void Element::set\_data(double information) { data = information; }
13. void Element::set\_data(string information) { spelling = information; }
14. void Element::set\_data(double data\_key, string information)
15. {
16. data = data\_key;
17. spelling = information;
18. }
19. Element::~Element() { delete next; next = nullptr; }

Stack.h:

1. #pragma once
2. #include "Element.h"
3. class Stack
4. {
5. private:
6. Element\* top;
7. void set\_top(Element\*);
8. public:
9. Stack();
10. bool is\_empty();
11. int size();
12. Element\* peek(); // to show who's on the top
13. Element\* pop(); // to delte top and show it
14. Element\* push(double); // to push on top and show it
15. void push(Element\*);
16. ~Stack();
17. };

Stack.cpp:

1. #include "Stack.h"
2. void Stack::set\_top(Element\* top\_element) { top = top\_element; }
3. Stack::Stack()
4. {
5. top = nullptr;
6. }
7. bool Stack::is\_empty()
8. {
9. if (top == nullptr)
10. return true;
11. return false;
12. }
13. int Stack::size()
14. {
15. int size = 1;
16. if (is\_empty())
17. return 0;
18. for (Element\* now = top; now->get\_next() != nullptr; now = now->get\_next())
19. size++;
20. return size;
21. }
22. Element\* Stack::peek() // to show who's on the top
23. {
24. return top;
25. }
26. Element\* Stack::pop() // to delte top and show it
27. {
28. Element\* to\_delete = top;
29. if (is\_empty())
30. throw out\_of\_range("the stack is empty");
31. else
32. top = top->get\_next();
33. return to\_delete;
34. }
35. Element\* Stack::push(double data) // to push on top and show it
36. {
37. Element\* new\_element = new Element;
38. new\_element->set\_data(data);
39. if (is\_empty())
40. top = new\_element;
41. else
42. {
43. new\_element->set\_next(top);
44. top = new\_element;
45. }
46. return new\_element;
47. }
48. void Stack::push(Element\* new\_element) // to push on top and show it
49. {
50. if (is\_empty())
51. top = new\_element;
52. else
53. {
54. new\_element->set\_next(top);
55. top = new\_element;
56. }
57. }
58. Stack::~Stack()
59. {
60. while (!is\_empty())
61. pop();
62. }

Queue.h:

1. #pragma once
2. #include "Element.h"
3. class Queue
4. {
5. private:
6. Element\* head; // the first in queue, leaves first
7. Element\* tail; // the last, new in queue
8. void set\_head(Element\*);
9. void set\_tail(Element\*);
10. public:
11. Queue();
12. int size(); // return a number of elements in queue
13. bool is\_empty(); // returns true if queue is empty
14. Element\* pop(); // to delete first in queue and return it to user
15. Element\* push(double); // to add in the end of the queue the element with key-data and return it to user
16. Element\* push(string);
17. Element\* push(double, string);
18. void push(Element\*);
19. //Element\* back(); // to know who is the last in queue and return it to user
20. ~Queue();
21. };

Queue.cpp:

1. #include "Queue.h"
2. Queue::Queue()
3. {
4. head = nullptr;
5. tail = nullptr;
6. }
7. void Queue::set\_head(Element\* key) { head = key; }
8. void Queue::set\_tail(Element\* key) { tail = key; }
9. bool Queue::is\_empty() // returns true if queue is empty
10. {
11. if (tail == nullptr)
12. return true;
13. return false;
14. }
15. int Queue::size() // return a number of elements in queue
16. {
17. if (is\_empty())
18. return 0;
19. int size = 1;
20. for (Element\* now = tail; now->get\_next() != nullptr; now = now->get\_next())
21. size++;
22. return size;
23. }
24. Element\* Queue::push(double data\_key) // to add in the end of the queue the element with key-data
25. {
26. Element\* new\_element = new Element;
27. new\_element->set\_data(data\_key);
28. if (is\_empty())
29. head = tail = new\_element;
30. else
31. {
32. new\_element->set\_next(tail);
33. tail = new\_element;
34. }
35. return new\_element;
36. }
37. Element\* Queue::push(string data) // to add in the end of the queue the element with key-data
38. {
39. Element\* new\_element = new Element;
40. new\_element->set\_data(data);
41. if (is\_empty())
42. head = tail = new\_element;
43. else
44. {
45. new\_element->set\_next(tail);
46. tail = new\_element;
47. }
48. return new\_element;
49. }
50. Element\* Queue::push(double number, string data) // to add in the end of the queue the element with key-data
51. {
52. Element\* new\_element = new Element;
53. new\_element->set\_data(number, data);
54. if (is\_empty())
55. head = tail = new\_element;
56. else
57. {
58. new\_element->set\_next(tail);
59. tail = new\_element;
60. }
61. return new\_element;
62. }
63. void Queue::push(Element\* new\_element) // to add in the end of the queue the element with key-data
64. {
65. if (is\_empty())
66. head = tail = new\_element;
67. else
68. {
69. new\_element->set\_next(tail);
70. tail = new\_element;
71. }
72. }
73. Element\* Queue::pop() // to delete first in queue and return it's key-data
74. {
75. Element\* to\_delete;
76. if (is\_empty())
77. throw out\_of\_range("the queue is empty");
78. else if (size() == 1)
79. {
80. to\_delete = head;
81. head = tail = nullptr;
82. }
83. else
84. {
85. to\_delete = head;
86. for (Element\* now = tail; now->get\_next() != nullptr; now = now->get\_next())
87. head = now;
88. head->next = nullptr;
89. }
90. return to\_delete;
91. }
92. Queue::~Queue()
93. {
94. while (!is\_empty())
95. pop();
96. }

Calculator.h:

1. #pragma once
2. #include "Queue.h"
3. #include "Stack.h"
4. #include <iostream>
5. #include <fstream>
6. #include <string>
7. #define \_USE\_MATH\_DEFINES // for pi written like M\_PI
8. #include <math.h>
9. enum functions { COS, SIN, TG, CTG, LN, LOG, SQRT, PI, EXP, ABS, NOT\_FUNCTION };
10. enum priorities { brecket, minus\_plus, multiply\_divide, degree, function, not\_operator };
11. extern bool my\_error;
12. bool is\_number(char);
13. bool is\_letter(char);
14. int what\_operation\_priority(char);
15. int what\_function(string);
16. Queue\* string\_processing(ifstream&, ofstream&);
17. Queue\* convert\_to\_reverse\_polish\_notation(Queue\*, ofstream&);
18. double calculation(ifstream&, ofstream&);

Calculator.cpp:

1. #include "Calculator.h"
2. bool my\_error = false; // definition
3. bool is\_number(char symbol)
4. {
5. string can\_begin\_with\_number("0123456789");
6. for (char now\_symbol : can\_begin\_with\_number)
7. {
8. if (symbol == now\_symbol)
9. return true;
10. }
11. return false;
12. }
13. bool is\_letter(char symbol)
14. {
15. string can\_be\_letter("cosintglqrtexpab");
16. for (char now\_symbol : can\_be\_letter)
17. {
18. if (symbol == now\_symbol)
19. return true;
20. }
21. return false;
22. }
23. int what\_operation\_priority(char symbol)
24. {
25. if (symbol == ')' || symbol == '(')
26. return brecket;
27. if (symbol == '+' || symbol == '-')
28. return minus\_plus;
29. if (symbol == '\*' || symbol == '/')
30. return multiply\_divide;
31. if (symbol == '^')
32. return degree;
33. return not\_operator;
34. }
35. int what\_function(string function)
36. {
37. if (function == "cos")
38. return COS;
39. if (function == "sin")
40. return SIN;
41. if (function == "tg")
42. return TG;
43. if (function == "ctg")
44. return CTG;
45. if (function == "ln")
46. return LN;
47. if (function == "log")
48. return LOG;
49. if (function == "sqrt")
50. return SQRT;
51. if (function == "pi")
52. return PI;
53. if (function == "exp")
54. return EXP;
55. if (function == "abs")
56. return ABS;
57. return NOT\_FUNCTION; // not a function of the list
58. }
59. double count\_function(int function, double argument)
60. { // можно выявить ошибки типо пи на два аргумента или деления на ноль
61. double counted = 0;
62. switch (function)
63. {
64. case COS:
65. counted = cos(argument);
66. break;
67. case SIN:
68. counted = sin(argument);
69. break;
70. case TG:
71. counted = tan(argument);
72. break;
73. case CTG:
74. counted = (cos(argument)/sin(argument));
75. break;
76. case LN:
77. counted = log(argument);
78. break;
79. case LOG:
80. counted = log10(argument);
81. break;
82. case SQRT:
83. counted = sqrt(argument);
84. break;
85. case ABS:
86. counted = abs(argument);
87. break;
88. case EXP:
89. counted = exp(argument);
90. break;
91. }
92. return counted;
93. }
94. double count\_operation(string operation, double first\_argument, double second\_argument)
95. {
96. double result = 0;
97. switch (operation[0])
98. {
99. case '-':
100. result = (first\_argument - second\_argument);
101. break;
102. case '+':
103. result = (first\_argument + second\_argument);
104. break;
105. case '\*':
106. result = (first\_argument \* second\_argument);
107. break;
108. case '/':
109. result = (first\_argument / second\_argument);
110. break;
111. case '^':
112. result = (pow(first\_argument, second\_argument));
113. break;
114. }
115. return result;
116. }
117. Queue\* string\_processing(ifstream& in, ofstream& out) // = 1 in start
118. {
119. char now\_char;
120. Queue\* elements = new Queue;
121. int left\_breckets = 0;
122. int right\_breckets = 0;
123. //out << "got input:" << endl;
124. while ((!my\_error) && (in.peek() != '\n') && (in.peek() != EOF))
125. {
126. if (in.peek() == ' ')
127. {
128. in.get();
129. continue;
130. }
131. in >> now\_char;
132. if (now\_char == '(') // if we met (
133. {
134. left\_breckets++;
135. elements->push(what\_operation\_priority(now\_char), "("); // add ( to queue
136. out << now\_char; // (
137. if (what\_operation\_priority(in.peek()) != not\_operator)
138. {
139. if (in.peek() == '-') //unary minus // ...(-...
140. elements->push(0, "unary\_minus"); // add null to have 0-A instead of just -A
141. else if (in.peek() != '(') // error ...(operator...
142. {
143. out << endl << "...don't know what does ( " << in.peek() << " mean " << endl;
144. my\_error = true;
145. }
146. }
147. }
148. else if (is\_number(now\_char)) // met a first digit of a number
149. {
150. double now\_number;
151. in.putback(now\_char);
152. in >> now\_number; // read whole number
153. elements->push(now\_number); // add number to queue (will have -1 priority (not set))
154. out << now\_number; // number
155. if (is\_letter(in.peek())) // error ...number letter...
156. {
157. out << endl << "...action is required after " << now\_number << endl;
158. my\_error = true;
159. }
160. else if (in.peek() == '(') // error ...number(
161. {
162. out << endl << "...don't know what does " << now\_number << "( mean" << endl;
163. my\_error = true;
164. }
165. }
166. else if (now\_char == ')') // met )
167. {
168. right\_breckets++;
169. if (right\_breckets > left\_breckets) // error like ...())...
170. {
171. out << endl << "...left brecket is required" << endl;
172. my\_error = true;
173. }
174. else if (in.peek() == '(') // error ...)(...
175. {
176. out << endl << "...don't know what does )( mean" << endl;
177. my\_error = true;
178. }
179. else if (is\_letter(in.peek())) // error ...)letter...
180. {
181. out << endl << "...action is required after )" << endl;
182. my\_error = true;
183. }
184. else
185. {
186. elements->push(what\_operation\_priority(now\_char), ")"); // push ) to queue
187. out << now\_char; // )
188. }
189. }
190. else if (what\_operation\_priority(now\_char) != not\_operator) // met an operator
191. {
192. if ((now\_char == '-')&&(elements->is\_empty()))
193. elements->push(0, "unary\_minus"); // add null to have 0-A instead of just -A
194. if (in.peek() == ')') // error ...operator)...
195. {
196. out << endl << "...operand is required after " << now\_char << endl;
197. my\_error = true;
198. }
199. else if ((what\_operation\_priority(in.peek()) != not\_operator)&&(in.peek() != '(')) // error ...operator operator(not ( )...
200. {
201. out << endl << "...don't know what does " << now\_char << " " << in.peek() << " mean" << endl;
202. my\_error = true;
203. }
204. else // no error // push operator to queue
205. {
206. string math\_operator;
207. math\_operator.push\_back(now\_char);
208. elements->push(what\_operation\_priority(now\_char), math\_operator); // operation with priority = 1/2/3
209. out << now\_char; //operator
210. }
211. }
212. else if (is\_letter(now\_char)) // met a letter
213. {
214. string function;
215. function.push\_back(now\_char); // want to have a word if it is cos for example
216. while (is\_letter(in.peek()))
217. {
218. in >> now\_char;
219. function.push\_back(now\_char);
220. } // now we have a word
221. while (in.peek() == ' ')
222. in.get();
223. if (what\_function(function) == PI) // pi doesn't need an argument
224. {
225. out << function;
226. elements->push(M\_PI, "pi"); // push pi
227. }
228. else if (what\_function(function) == NOT\_FUNCTION) // error: if we don't know this word
229. {
230. out << endl << "...don't know what is " << function << endl;
231. my\_error = true;
232. }
233. else if (in.peek() != '(') // error ...function... (cos5 should be cos(5))
234. {
235. out << endl << "...no argument for " << function << endl;
236. my\_error = true;
237. }
238. else
239. {
240. elements->push(priorities::function, function); // push function to queue
241. out << function; // function
242. }
243. }
244. else
245. { // error: unknown symbol (maybe not situated like . )
246. out << endl << "...don't know what does " << now\_char << " mean here" << endl;
247. my\_error = true;
248. }
249. }
250. if ((right\_breckets < left\_breckets)&&(!my\_error)) // error like ...(() EOF
251. {
252. out << endl << "...there is missing one or more right breckets" << endl;
253. my\_error = true;
254. }
255. if (my\_error)
256. {
257. out << "try again..." << endl;
258. return nullptr;
259. }
260. return elements;
261. }
262. Queue\* convert\_to\_reverse\_polish\_notation(Queue\* elements, ofstream& out)
263. {
264. Queue\* reversed = new Queue;
265. if (elements == nullptr)
266. return nullptr;
267. Stack stack;
268. while (!elements->is\_empty())
269. {
270. Element\* now\_element = new Element;
271. now\_element = elements->pop();
272. string now\_string = now\_element->get\_spelling();
273. double now\_data = now\_element->get\_data();
274. if (now\_string == "(")
275. stack.push(now\_element);
276. else if ((now\_string == "#") || (now\_string == "unary\_minus") || (now\_string == "pi")) // operand
277. {
278. out << now\_data << " ";
279. reversed->push(now\_element);
280. }
281. else if (now\_string == ")")
282. {
283. while (stack.peek()->get\_spelling() != "(")
284. {
285. out << stack.peek()->get\_spelling() << " ";
286. reversed->push(stack.pop());
287. }
288. stack.pop(); // (
289. }
290. else // operator or function
291. {
292. if (!stack.is\_empty())
293. {
294. while (now\_data <= stack.peek()->get\_data())// comparing priorities
295. {
296. out << stack.peek()->get\_spelling() << " ";
297. reversed->push(stack.pop());
298. if (stack.is\_empty())
299. break;
300. }
301. }
302. stack.push(now\_element);
303. }
304. }
305. while (!stack.is\_empty())
306. {
307. out << stack.peek()->get\_spelling();
308. reversed->push(stack.pop());
309. }
310. return reversed;
311. }
312. double calculation(ifstream& in, ofstream& out)
313. {
314. out << "got input:" << endl;
315. Queue\* elements = string\_processing(in, out);
316. if ((elements == nullptr)&&(!my\_error))
317. {
318. out << "error: no elements" << endl;
319. my\_error = true;
320. return 0;
321. }
322. if (my\_error)
323. return 0;
324. out << endl << "reverse polish notation of input:" << endl;
325. elements = convert\_to\_reverse\_polish\_notation(elements, out);
326. if ((elements == nullptr) && (!my\_error))
327. {
328. out << "error: no reversed elements" << endl;
329. my\_error = true;
330. return 0;
331. }
332. double result = 0;
333. if (my\_error)
334. return 0;
335. Stack stack;
336. while ((!elements->is\_empty())&&(!my\_error))
337. {
338. Element\* now\_element = elements->pop();
339. string now\_string = now\_element->get\_spelling();
340. double now\_data = now\_element->get\_data();
341. if ((now\_string == "#") || (now\_string == "unary\_minus") || (now\_string == "pi"))
342. {
343. stack.push(now\_element); // operand
344. }
345. else // function or operator
346. {
347. double first\_operand = 0;
348. if ((now\_data == priorities::function) && (!stack.is\_empty()))
349. {
350. result = count\_function(what\_function(now\_string), stack.pop()->get\_data());
351. stack.push(result);
352. }
353. else if (!stack.is\_empty()) //ИНАЧЕ
354. {
355. result = stack.pop()->get\_data(); // second operand
356. if (stack.is\_empty())
357. {
358. out << endl << "error: not enough operands" << endl;
359. out << now\_data << endl;
360. my\_error = true;
361. return 0;
362. }
363. first\_operand = stack.pop()->get\_data();
364. result = count\_operation(now\_string, first\_operand, result);
365. stack.push(result);
366. }
367. }
368. }
369. return result;
370. }

CalculatorTest.cpp:

1. #include "pch.h"
2. #include "CppUnitTest.h"
3. #include "/Users/varka/source/repos/Calculator/Calculator.h"
4. using namespace Microsoft::VisualStudio::CppUnitTestFramework;
5. //extern bool my\_error;
6. namespace CalculatorTest
7. {
8. TEST\_CLASS(CalculatorTest)
9. {
10. public:
11. TEST\_METHOD(test\_error\_breckets)
12. {
13. my\_error = false;
14. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
15. out << "cos((1-(2+3))\*4)^(5+6)";
16. out.close();
17. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
18. out.open("/Users/varka/source/repos/Calculator/out.txt");
19. double result = calculation(in, out);
20. out.close();
21. Assert::IsFalse(my\_error);
22. }
23. TEST\_METHOD(test\_error\_unknown\_symbol)
24. {
25. my\_error = false;
26. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
27. out << "cos((1.2534.173-(2+3))\*4)^(5+6))";
28. out.close();
29. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
30. out.open("/Users/varka/source/repos/Calculator/out.txt");
31. double result = calculation(in, out);
32. out.close();
33. Assert::IsTrue(my\_error);
34. }
35. TEST\_METHOD(test\_error\_double\_operator)
36. {
37. my\_error = false;
38. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
39. out << "cos(1-(2++3))\*4^(5+6)";
40. out.close();
41. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
42. out.open("/Users/varka/source/repos/Calculator/out.txt");
43. double result = calculation(in, out);
44. out.close();
45. Assert::IsTrue(my\_error);
46. }
47. TEST\_METHOD(test\_error\_no\_operator)
48. {
49. my\_error = false;
50. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
51. out << "cos(1(2+3))\*4^(5+6)";
52. out.close();
53. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
54. out.open("/Users/varka/source/repos/Calculator/out.txt");
55. double result = calculation(in, out);
56. out.close();
57. Assert::IsTrue(my\_error);
58. }
59. TEST\_METHOD(readInput)
60. {
61. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
62. out << "(cos((1-(2+3))\*4))^(5+6)";
63. out.close();
64. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
65. out.open("/Users/varka/source/repos/Calculator/out.txt");
66. Queue\* elements = string\_processing(in, out);
67. in.close();
68. out.close();
69. in.open("/Users/varka/source/repos/Calculator/in.txt");
70. while ((in.peek() != '\n') && (in.peek() != EOF))
71. {
72. while (in.peek() == ' ')
73. in.get();
74. if (elements->is\_empty())
75. Assert::Fail();
76. Element\* element = elements->pop();
77. string function = element->get\_spelling();
78. double data = element->get\_data();
79. if (function == "unary\_minus")
80. continue;
81. if (function != "#")
82. {
83. char now\_char;
84. for (unsigned int i = 0; i < function.size(); i++)
85. {
86. in >> now\_char;
87. char in\_string = function[i];
88. Assert::AreEqual(in\_string, now\_char);
89. while ((in.peek() == ' ') && (in.peek() != '\n') && (in.peek() != EOF))
90. in.get();
91. }
92. }
93. else
94. {
95. if (in.peek() == 'p') // pi case
96. {
97. in.get();
98. if (in.peek() == 'i')
99. {
100. in.get();
101. Assert::AreEqual(data, M\_PI);
102. }
103. else
104. in.putback('p');
105. }
106. else
107. {
108. double number;
109. in >> number;
110. Assert::AreEqual(data, number);
111. }
112. }
113. }
114. Assert::IsTrue(true);
115. }
116. TEST\_METHOD(test\_convert\_to\_reverse\_polish\_notation)
117. {
118. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
119. out << "(cos((1-(2+3))\*4))^(5+6)";
120. string right\_result = "1 2 3 + - 4 \* cos 5 6 + ^";
121. out.close();
122. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
123. out.open("/Users/varka/source/repos/Calculator/out.txt");
124. Queue\* elements = string\_processing(in, out);
125. in.close();
126. out.close();
127. out.open("/Users/varka/source/repos/Calculator/out.txt");
128. convert\_to\_reverse\_polish\_notation(elements, out);
129. out.close();
130. in.open("/Users/varka/source/repos/Calculator/out.txt");
131. string result;
132. string current;
133. while ((in.peek() != '\n') && (in.peek() != EOF))
134. {
135. in >> current;
136. result += " " + current;
137. }
138. result.erase(0, 1);
139. in.close();
140. Assert::AreEqual(result, right\_result);
141. }
142. TEST\_METHOD(test\_count)
143. {
144. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
145. out << "(cos((1-(2+3))\*4))^(5+6)";
146. out.close();
147. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
148. out.open("/Users/varka/source/repos/Calculator/out.txt");
149. double result = calculation(in, out);
150. out << endl << "result = " << result << endl;
151. double right\_result = pow(cos(-16), 11);
152. out.close();
153. Assert::AreEqual(result, right\_result);
154. }
155. TEST\_METHOD(test\_1\_count)
156. {
157. my\_error = false;
158. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
159. out << "-cos(3)";
160. out.close();
161. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
162. out.open("/Users/varka/source/repos/Calculator/out.txt");
163. double result = calculation(in, out);
164. out << endl << "result = " << result << endl;
165. double right\_result = -cos(3);
166. out.close();
167. Assert::AreEqual(result, right\_result);
168. }
169. TEST\_METHOD(test\_2\_count)
170. {
171. my\_error = false;
172. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
173. out << "ln(23.1278\*exp(-4))";
174. out.close();
175. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
176. out.open("/Users/varka/source/repos/Calculator/out.txt");
177. double result = calculation(in, out);
178. out << endl << "result = " << result << endl;
179. double right\_result = log(23.1278 \* exp(-4));
180. out.close();
181. Assert::AreEqual(result, right\_result);
182. }
183. TEST\_METHOD(test\_3\_count)
184. {
185. my\_error = false;
186. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
187. out << "tg(log(abs(ln(23.1278\*exp(-4)))))";
188. out.close();
189. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
190. out.open("/Users/varka/source/repos/Calculator/out.txt");
191. double result = calculation(in, out);
192. out << endl << "result = " << result << endl;
193. double right\_result = tan(log10(abs(log(23.1278 \* exp(-4)))));
194. out.close();
195. Assert::AreEqual(result, right\_result);
196. }
197. TEST\_METHOD(test\_4\_count)
198. {
199. my\_error = false;
200. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
201. out << "cos(pi)\*tg(log(abs(ln(23.1278\*exp(-4)))))/13.0498-sin(3^7)";
202. out.close();
203. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
204. out.open("/Users/varka/source/repos/Calculator/out.txt");
205. double result = calculation(in, out);
206. out << endl << "result = " << result << endl;
207. double right\_result = cos(M\_PI) \* tan(log10(abs(log(23.1278 \* exp(-4))))) / 13.0498 - sin(pow(3, 7));
208. out.close();
209. Assert::AreEqual(result, right\_result);
210. }
211. TEST\_METHOD(test\_5\_count)
212. {
213. my\_error = false;
214. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
215. out << "-cos(-pi+1)\*tg(log(abs(ln(23.1278\*exp(-4)))))/13.0498-sin(3^7)";
216. out.close();
217. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
218. out.open("/Users/varka/source/repos/Calculator/out.txt");
219. double result = calculation(in, out);
220. out << endl << "result = " << result << endl;
221. double right\_result = -cos(-M\_PI+1) \* tan(log10(abs(log(23.1278 \* exp(-4))))) / 13.0498 - sin(pow(3, 7));
222. out.close();
223. Assert::AreEqual(result, right\_result);
224. }
225. TEST\_METHOD(test\_6\_count)
226. {
227. my\_error = false;
228. ofstream out("/Users/varka/source/repos/Calculator/in.txt");
229. out << "cos( pi) \* tg(- log(abs(-ln(23.1278\* exp(-4))))) / 13.0498-sin(3^ 7) ";
230. out.close();
231. ifstream in("/Users/varka/source/repos/Calculator/in.txt");
232. out.open("/Users/varka/source/repos/Calculator/out.txt");
233. double result = calculation(in, out);
234. out << endl << "result = " << result << endl;
235. double right\_result = cos(M\_PI) \* tan(-log10(abs(-log(23.1278 \* exp(-4))))) / 13.0498 - sin(pow(3, 7));
236. out.close();
237. Assert::AreEqual(result, right\_result);
238. }
239. };
240. }