Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение

высшего образования

«**Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные

системы»

**ОТЧЁТ**

**по творческой работе**

Темы: «Калькулятор. Хранение капиталов в банке.

Решение задачи коммивояжёра.»

Вариант 17

Выполнил:

Студент группы ИВТ-20-2б

Сафронов Владислав Владиславович

Проверила:

Доцент кафедры ИТАС

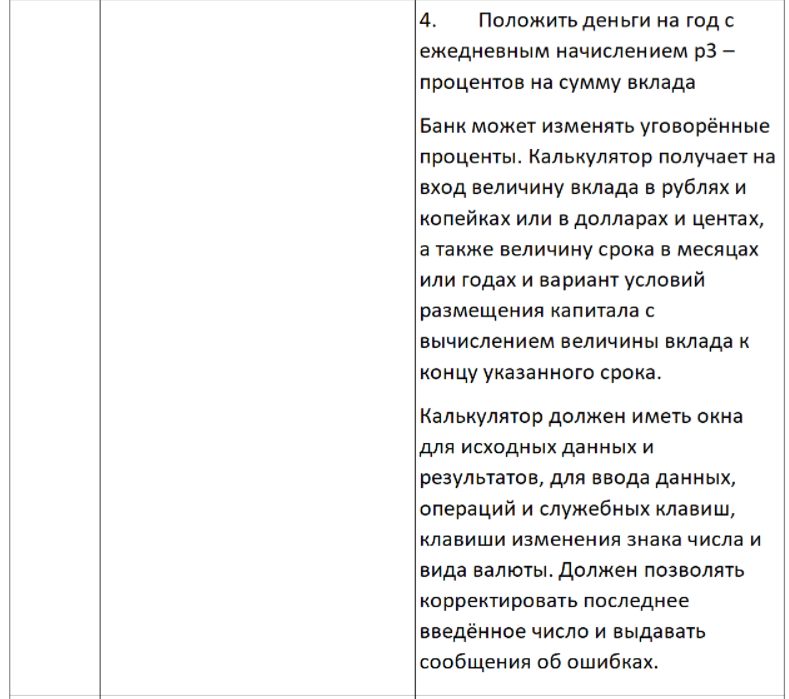
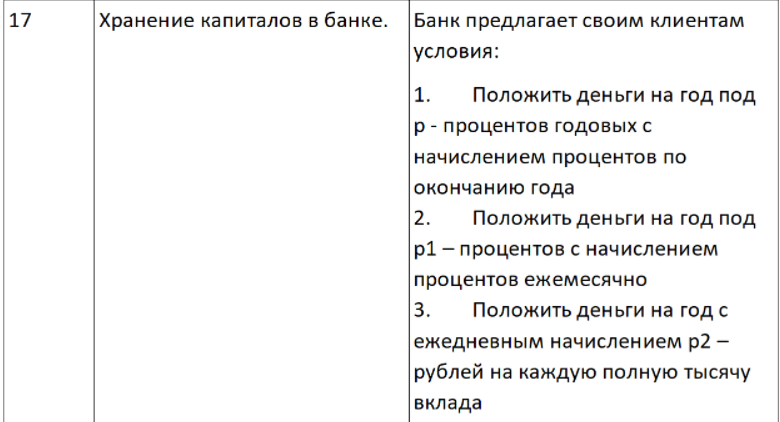
Полякова О.А.

**Пермь, 2021**

**Разработка калькулятора**

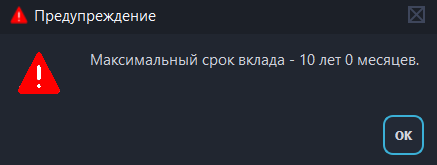
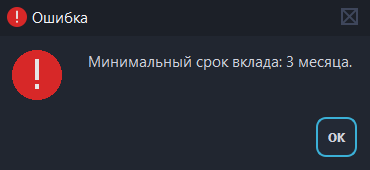
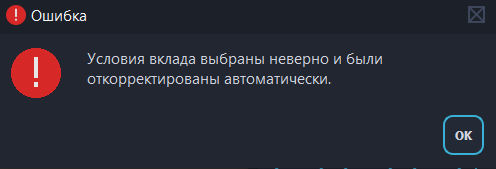
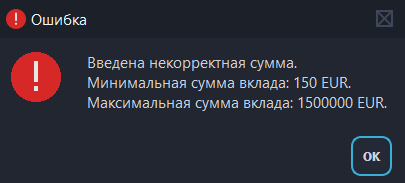
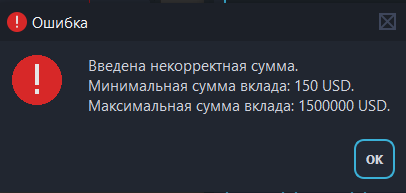
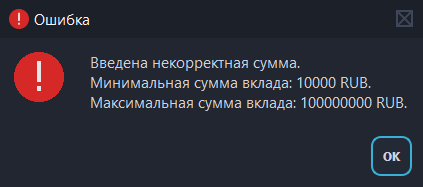
**Постановка задачи**

1. Разработать алгоритм калькулятора в соответствии с заданием.
2. Реализовать алгоритм в виде программы на алгоритмическом языке C++.
3. Разработать интерфейс средствами Qt.



**Анализ задачи**

1. Описание алгоритма работы калькулятора.
   1. Ввод данных. В форме MainWindow для получения пользовательских данных на форме присутствуют кнопки (класс QPushButton). Также имеется поддержка ввода с клавиатуры. Введенные данные отображаются на поле QLabel. Для изменения срока вклада был использованы 2 поля QSpinBox. Для выбора условий вклада и валюты были реализованы поля QComboBox. При изменении срока вклада условия автоматически изменяются на максимально выгодное из доступных. В форме SettingsForm были реализованы поля QDoubleSpinBox для изменения процентных ставок по каждому условию и изменения курсов валют.
   2. В случае неправильного ввода реализован вывод ошибок с помощью класса QMessageBox.



* 1. Вывод данных. В форме ResultForm расположено 3 поля для вывода данных: QLabel для вывода полученной суммы по окончанию срока, QLabel для вывода изначальной суммы вклада и QLabel для вывода суммы процентов, которые выплатит банк.

1. – 5. Описание структуры программы.
   1. Класс MainWindow наследуется от класса QMainWindow. В классе объявлены переменные для хранения актуальных данных о процентных ставках, выбранной валюте и курсах валют.
2. public:
3. MainWindow(QWidget \*parent = nullptr);
4. ~MainWindow();
6. int current\_value = 0;
7. double p = 5, p1 = 3.2, p2 = 3, p3 = 2.6;
8. double RUB\_TO\_USD = 74.0;
9. double RUB\_TO\_EUR = 89.0;
   1. В классе реализованы необходимые слоты и методы.
10. private slots:
11. void on\_button\_c\_clicked();
12. void add\_digit();
13. void show\_percent();
14. void convert\_value();
16. void on\_button\_point\_clicked();
18. void on\_spin\_mon\_valueChanged();
20. void on\_spin\_yr\_valueChanged();
22. void on\_button\_calc\_clicked();
24. void on\_button\_help\_clicked();
26. void on\_button\_settings\_clicked();
28. void on\_button\_erase\_clicked();
30. private:
31. Ui::MainWindow \*ui;
33. double calculate();
34. void check\_terms();
35. bool check\_right\_summ();
36. bool check\_right\_time();

* 1. Метод check\_terms используется для автоматического изменения условий вклада при изменении срока вклада.

1. void MainWindow::check\_terms() //изменение условий при изменении срока вклада
2. {
3. int mn = ui->spin\_mon->value() + (ui->spin\_yr->value() \* 12);
4. if (mn < 6) {
5. ui->combo\_terms->setCurrentIndex(3);
6. } else
7. if (mn >= 6 && mn < 12) {
8. ui->combo\_terms->setCurrentIndex(2);
9. } else
10. if (mn >= 12 && mn < 36) {
11. ui->combo\_terms->setCurrentIndex(1);
12. } else
13. if (mn >= 36) {
14. ui->combo\_terms->setCurrentIndex(0);
15. }
16. }
    1. Метод convert\_value разработан для конвертирования валют. Какая валюта была установлена до изменения. Если это доллары или евро, то сначала введенная сумма переводится в рубли. Затем происходит перевод в ту валюту, которую выбрал пользователь.
17. void MainWindow::convert\_value() //конвертация валют
18. {
19. QString new\_lbl;
20. double tmp = ui->result\_show->text().toDouble();
21. if (current\_value == 1) {
22. tmp \*= RUB\_TO\_USD;
23. } else
24. if (current\_value == 2) {
25. tmp \*= RUB\_TO\_EUR;
26. }
28. if (ui->combo\_value->currentIndex() == 1) {
29. tmp /= RUB\_TO\_USD;
30. } else
31. if (ui->combo\_value->currentIndex() == 2) {
32. tmp /= RUB\_TO\_EUR;
33. }
34. tmp = floor(tmp \* 10) / 10;
36. current\_value = ui->combo\_value->currentIndex();
38. new\_lbl = QString::number(tmp, 'g', 10);
39. ui->result\_show->setText(new\_lbl);
40. }
    1. Метод check\_right\_summ используется для проверки корректности введенной суммы вклада. Если введенная сумма не соответствует условию, то выводится сообщение об ошибке. Минимальная и максимальная суммы вкладов для каждой валюты своя.
41. bool MainWindow::check\_right\_summ() //проверка корректности введенной суммы вклада
42. {
43. double tmp = ui->result\_show->text().toDouble();
44. double MIN\_SUM, MAX\_SUM;
45. switch (current\_value) {
46. case 0: {
47. MIN\_SUM = 10000;
48. MAX\_SUM = 100000000;
49. if (tmp < MIN\_SUM || tmp > MAX\_SUM) {
50. QMessageBox msgBox;
51. QPixmap icon(":/icons/icons/error.png");
52. msgBox.setWindowTitle("Ошибка");
53. msgBox.setText("Введена некорректная сумма. \nМинимальная сумма вклада: 10000 RUB.\nМаксимальная сумма вклада: 100000000 RUB.");
54. msgBox.setWindowIcon(icon);
55. msgBox.setIconPixmap(icon.scaled(50, 50));
56. msgBox.exec();
57. return false;
58. }
59. break;
60. }
61. case 1: {
62. MIN\_SUM = 150;
63. MAX\_SUM = 1500000;
64. if (tmp < MIN\_SUM || tmp > MAX\_SUM) {
65. QMessageBox msgBox;
66. QPixmap icon(":/icons/icons/error.png");
67. msgBox.setWindowTitle("Ошибка");
68. msgBox.setText("Введена некорректная сумма. \nМинимальная сумма вклада: 150 USD.\nМаксимальная сумма вклада: 1500000 USD.");
69. msgBox.setWindowIcon(icon);
70. msgBox.setIconPixmap(icon.scaled(50, 50));
71. msgBox.exec();
72. return false;
73. }
74. break;
75. }
76. case 2:{
77. MIN\_SUM = 150;
78. MAX\_SUM = 1500000;
79. if (tmp < MIN\_SUM || tmp > MAX\_SUM) {
80. QMessageBox msgBox;
81. QPixmap icon(":/icons/icons/error.png");
82. msgBox.setWindowTitle("Ошибка");
83. msgBox.setText("Введена некорректная сумма. \nМинимальная сумма вклада: 150 EUR.\nМаксимальная сумма вклада: 1500000 EUR.");
84. msgBox.setWindowIcon(icon);
85. msgBox.setIconPixmap(icon.scaled(50, 50));
86. msgBox.exec();
87. return false;
88. }
89. break;
90. }
91. }
92. return true;
93. }
    1. Метод check\_right\_time используется для проверки правильности выбранных условий вклада в соответствии с введенным сроком. Если условия выбраны неверно, то выводится сообщение об ошибке и автоматически выбирается наиболее выгодное корректное условие вклада.
94. bool MainWindow::check\_right\_time() //проверка правильности выбранных условий вклада
95. {
96. int index = ui->combo\_terms->currentIndex();
97. int mn = ui->spin\_mon->value() + (ui->spin\_yr->value() \* 12);
98. if (mn < 3) {
99. QMessageBox msgBox;
100. QPixmap icon(":/icons/icons/error.png");
101. msgBox.setWindowTitle("Ошибка");
102. msgBox.setText("Минимальный срок вклада: 3 месяца.");
103. msgBox.setWindowIcon(icon);
104. msgBox.setIconPixmap(icon.scaled(50, 50));
105. msgBox.exec();
106. return false;
107. } else
108. if (mn < 6 && (index == 2 || index == 1 || index == 0)) {
109. QMessageBox msgBox;
110. QPixmap icon(":/icons/icons/error.png");
111. msgBox.setWindowTitle("Ошибка");
112. msgBox.setText("Условия вклада выбраны неверно и были откорректированы автоматически.");
113. msgBox.setWindowIcon(icon);
114. msgBox.setIconPixmap(icon.scaled(50, 50));
115. msgBox.exec();
116. check\_terms();
117. return false;
118. } else
119. if (mn < 12 && (index == 1 || index == 0)) {
120. QMessageBox msgBox;
121. QPixmap icon(":/icons/icons/error.png");
122. msgBox.setWindowTitle("Ошибка");
123. msgBox.setText("Условия вклада выбраны неверно и были откорректированы автоматически.");
124. msgBox.setWindowIcon(icon);
125. msgBox.setIconPixmap(icon.scaled(50, 50));
126. msgBox.exec();
127. check\_terms();
128. return false;
129. } else
130. if (mn < 36 && (index == 0)) {
131. QMessageBox msgBox;
132. QPixmap icon(":/icons/icons/error.png");
133. msgBox.setWindowTitle("Ошибка");
134. msgBox.setText("Условия вклада выбраны неверно и были откорректированы автоматически.");
135. msgBox.setWindowIcon(icon);
136. msgBox.setIconPixmap(icon.scaled(50, 50));
137. msgBox.exec();
138. check\_terms();
139. return false;
140. } else return true;
141. }
     1. Метод calculate рассчитывает результат вклада в зависимости от выбранных условий и срока вклада.
142. double MainWindow::calculate() //рассчет результата
143. {
144. double tmp = ui->result\_show->text().toDouble();
146. switch (ui->combo\_terms->currentIndex()) {
147. case 0: {
148. int yrs = ui->spin\_yr->value();
149. for (int i = 0; i < yrs;i++)
150. {
151. tmp = tmp + (tmp \* (0.01 \* p));
152. } break;
153. }
154. case 1: {
155. int months = ui->spin\_mon->value() + (ui->spin\_yr->value() \* 12);
156. for (int i = 0; i < months; i++)
157. {
158. tmp = tmp + (tmp \* (0.01 \* (p1 / 12.0)));
159. } break;
160. }
161. case 2: {
162. int days = (ui->spin\_mon->value() \* 30) + (ui->spin\_yr->value() \* 365);
163. for (int i = 0; i < days; i++)
164. {
165. tmp = tmp + (((double)((int)tmp / 1000) \* 1000) \* ( 0.01 \* (p2 / 365.0)));
166. } break;
167. }
168. case 3: {
169. int days = (ui->spin\_mon->value() \* 30) + (ui->spin\_yr->value() \* 365);
170. for (int i = 0; i < days; i++)
171. {
172. tmp = tmp + (tmp \* (0.01 \* (3 / 365.0)));
173. } break;
174. }
175. }
176. return tmp;
177. }
     1. Слот on\_button\_c\_clicked заменяет введенную сумму на 0.
178. void MainWindow::on\_button\_c\_clicked() //стереть всё
179. {
180. ui->result\_show->setText("0");
181. }
     1. Слот on\_button\_erase\_clicked стирает последний введенный символ.
182. void MainWindow::on\_button\_erase\_clicked() //стереть последний элемент
183. {
184. QString new\_lbl = ui->result\_show->text();
185. if (new\_lbl != "0") {
186. new\_lbl.chop(1);
187. ui->result\_show->setText(new\_lbl);
188. }
189. }
     1. Слот add\_digit осуществляет ввод цифр.
190. void MainWindow::add\_digit() //ввод цифр
191. {
192. double numbers;
193. QString new\_lbl;
194. QPushButton \*button = (QPushButton\*)sender();
196. if (ui->result\_show->text().contains(".") && button->text() == "0") {
197. new\_lbl = ui->result\_show->text() + button->text();
198. } else {
199. numbers = (ui ->result\_show->text() + button->text()).toDouble();
200. new\_lbl = QString::number(numbers, 'g', 10);
201. }
202. ui->result\_show->setText(new\_lbl);
203. }
     1. Слот on\_button\_point\_clicked проверяет, добавляет точку в случае, если поле еще не содержит точки.
204. void MainWindow::on\_button\_point\_clicked() //ввод точки
205. {
206. if(!(ui->result\_show->text().contains('.')))
207. ui->result\_show->setText(ui->result\_show->text() + ".");
208. }
     1. Слоты on\_spin\_mon\_valueChanged и on\_spin\_yr\_valueChanged отвечают за изменение срока вклада. Если срок вклада превысит максимальный, выводится соответствующая ошибка. Если количество месяцев равно 12, количество месяцев обнуляется и к полю spin\_yr прибавляется 1. Если количество месяцев равно -1, то количество месяцев становится равным 11, а из количества лет вычитается 1.
209. void MainWindow::on\_spin\_mon\_valueChanged()
210. {
211. if (ui->spin\_mon->value() == 12 && ui->spin\_yr->value() < 10)
212. {
213. ui->spin\_yr->setValue(ui->spin\_yr->value() + 1);
214. ui->spin\_mon->setValue(0);
215. } else
216. if (ui->spin\_mon->value() == -1 && ui->spin\_yr->value() > 0)
217. {
218. ui->spin\_yr->setValue(ui->spin\_yr->value() - 1);
219. ui->spin\_mon->setValue(11);
220. } else
221. if (ui->spin\_mon->value() == -1 && ui->spin\_yr->value() == 0)
222. {
223. ui->spin\_mon->setValue(0);
224. } else
225. if (ui->spin\_yr->value() == 10 && ui->spin\_mon->value() > 0)
226. {
227. QMessageBox msgBox;
228. QPixmap icon(":/icons/icons/warning.png");
229. msgBox.setWindowTitle("Предупреждение");
230. msgBox.setText("Максимальный срок вклада - 10 лет 0 месяцев.");
231. msgBox.setWindowIcon(icon);
232. msgBox.setIconPixmap(icon.scaled(50, 50));
233. msgBox.exec();
234. //QMessageBox::warning(this, "Предупреждение", "Максимальный срок вклада - 10 лет 0 месяцев.");
235. ui->spin\_mon->setValue(0);
236. }
237. check\_terms();
238. }
240. void MainWindow::on\_spin\_yr\_valueChanged()
241. {
242. if (ui->spin\_yr->value() == 11)
243. {
244. QMessageBox msgBox;
245. QPixmap icon(":/icons/icons/warning.png");
246. msgBox.setWindowTitle("Предупреждение");
247. msgBox.setText("Максимальный срок вклада - 10 лет 0 месяцев.");
248. msgBox.setWindowIcon(icon);
249. msgBox.setIconPixmap(icon.scaled(50, 50));
250. msgBox.exec();
251. //QMessageBox::warning(this, "Предупреждение", "Максимальный срок вклада - 10 лет 0 месяцев.");
253. ui->spin\_yr->setValue(10);
254. ui->spin\_mon->setValue(0);
255. } else
256. if (ui->spin\_yr->value() == 10)
257. {
258. ui->spin\_mon->setValue(0);
259. }
260. check\_terms();
261. }
     1. Метод show\_percent отвечает за отображение в окне актуальной процентной ставки по вкладу в зависимости от выбранных условий.
262. void MainWindow::show\_percent() //отображение процентной ставки
263. {
264. switch (ui->combo\_terms->currentIndex()) {
265. case 0: {
266. QString new\_lbl;
267. new\_lbl = QString::number(p, 'g', 10);
268. ui->percent\_show->setText(new\_lbl + "%");
269. break;
270. }
271. case 1: {
272. QString new\_lbl;
273. new\_lbl = QString::number(p1, 'g', 10);
274. ui->percent\_show->setText(new\_lbl + "%");
275. break;
276. }
277. case 2: {
278. QString new\_lbl;
279. new\_lbl = QString::number(p2, 'g', 10);
280. ui->percent\_show->setText(new\_lbl + "%");
281. break;
282. }
283. case 3: {
284. QString new\_lbl;
285. new\_lbl = QString::number(p3, 'g', 10);
286. ui->percent\_show->setText(new\_lbl + "%");
287. break;
288. }
289. }
290. }
     1. Слот on\_button\_calc\_clicked по нажатию кнопки расчета производит проверку правильности введенных данных с помощью функций check\_right\_time и check\_right\_summ и, в случае, если ввод корректен, производит подсчет результата с помощью функции calculate и выводит результат в окне ResultForm.
291. void MainWindow::on\_button\_calc\_clicked() //рассчет и вывод результата
292. {
293. double old;
294. if (check\_right\_time() && check\_right\_summ()) {
295. double diff = old = ui->result\_show->text().toDouble();
297. double result = calculate();
299. result = floor(result \* 10) / 10;
301. diff = result - diff;
302. diff = floor(diff \* 10) / 10;
304. ResultForm sec\_win;
305. sec\_win.setWindowFlags (sec\_win.windowFlags() & ~Qt::WindowContextHelpButtonHint);
306. sec\_win.setModal(true);
307. sec\_win.old\_sum = old;
308. sec\_win.difference = diff;
309. sec\_win.new\_sum = result;
310. sec\_win.value = ui->combo\_value->currentText();
311. sec\_win.show\_result();
312. sec\_win.exec();
313. }
314. }
     1. Слот on\_button\_help\_clicked по нажатию на кнопку открывает окно помощи с информацией о всех условиях вклада, минимальных и максимальных сроках вклада по всем условиям, а также максимальную и минимальную суммы вкладов для каждой из валют.
315. void MainWindow::on\_button\_help\_clicked() //вывод окна-подсказки
316. {
317. AboutForm about;
318. about.setWindowFlags (about.windowFlags() & ~Qt::WindowContextHelpButtonHint);
319. about.setModal(true);
320. about.fill\_percents(p, p1, p2, p3);
321. about.exec();
322. }
     1. Слот on\_button\_settings\_clicked по нажатию на кнопку открывает окно настроек с возможностью изменения процентной ставки по каждому из условий, а также корректировать курсы валют.
323. void MainWindow::on\_button\_settings\_clicked() //вывод окна с настройками
324. {
325. SettingsForm settings;
326. settings.setWindowFlags (settings.windowFlags() & ~Qt::WindowContextHelpButtonHint);
327. settings.setModal(true);
328. settings.old\_p = settings.p = p;
329. settings.old\_p1 = settings.p1 = p1;
330. settings.old\_p2 = settings.p2 = p2;
331. settings.old\_p3 = settings.p3 = p3;
332. settings.old\_usd = settings.usd = RUB\_TO\_USD;
333. settings.old\_eur = settings.eur = RUB\_TO\_EUR;
334. settings.check\_percents();
335. settings.exec();
336. if (settings.is\_saved) {
337. p = settings.p;
338. p1 = settings.p1;
339. p2 = settings.p2;
340. p3 = settings.p3;
341. RUB\_TO\_USD = settings.usd;
342. RUB\_TO\_EUR = settings.eur;
343. }
344. show\_percent();
345. }
346. Класс AboutForm наследуется от класса QDialog и имеет единственный метод fill\_percents.
347. class AboutForm : public QDialog
348. {
349. Q\_OBJECT
351. public:
352. explicit AboutForm(QWidget \*parent = nullptr);
353. ~AboutForm();
355. void fill\_percents(double p1, double p2, double p3, double p4);
357. private:
358. Ui::AboutForm \*ui;
359. };
     1. Метод fill\_percents используется для отображения актуальной информации о процентных ставках по каждому условию вклада.
360. void AboutForm::fill\_percents(double p1, double p2, double p3, double p4)
361. {
362. QString new\_lbl;
364. new\_lbl = ui->label\_term\_1->text() + QString::number(p1, 'g', 10);
365. ui->label\_term\_1->setText(new\_lbl + "%.");
367. new\_lbl = ui->label\_term\_2->text() + QString::number(p2, 'g', 10);
368. ui->label\_term\_2->setText(new\_lbl + "%.");
370. new\_lbl = ui->label\_term\_3->text() + QString::number(p3, 'g', 10);
371. ui->label\_term\_3->setText(new\_lbl + "%.");
373. new\_lbl = ui->label\_term\_4->text() + QString::number(p4, 'g', 10);
374. ui->label\_term\_4->setText(new\_lbl + "%.");
375. }
376. Класс ResultForm наследуется от класса QDialog, имеет поля old\_sum, new\_sum и difference для хранения введенной суммы, итоговой суммы и суммы процентов к выплате соответственно. Также класс имеет поле value для хранения выбранной валюты, метод show\_result и слот on\_pushButton\_clicked.
377. class ResultForm : public QDialog
378. {
379. Q\_OBJECT
381. public:
382. explicit ResultForm(QWidget \*parent = nullptr);
383. ~ResultForm();
385. double old\_sum, new\_sum, difference;
386. QString value;
387. void show\_result();
389. private slots:
390. void on\_pushButton\_clicked();
392. private:
393. Ui::ResultForm \*ui;
394. };
395. 1. Метод show\_result используется для конвертирования чисел в строки и передачи их в соответствующие поля на окне.
396. void ResultForm::show\_result()
397. {
398. QString new\_lbl;
400. new\_lbl = QString::number(new\_sum, 'g', 10);
401. ui->label\_res->setText(new\_lbl);
403. new\_lbl = QString::number(old\_sum, 'g', 10);
404. ui->label\_old->setText(new\_lbl);
406. new\_lbl = QString::number(difference, 'g', 10);
407. ui->label\_difference->setText(new\_lbl);
409. ui->label\_value\_1->setText(value);
410. ui->label\_value\_2->setText(value);
411. ui->label\_value\_3->setText(value);
412. }
     1. Слот on\_pushButton\_clicked по нажатию на кнопку закрывает окно.
413. void ResultForm::on\_pushButton\_clicked()
414. {
415. close();
416. }
417. Класс SettingsForm наследуется от класса QDialog. Класс имеет вещественные переменные для хранения старых и новых процентных ставок и валют, а также булевую переменную is\_saved. Также в классе реализовны необходимые методы и слоты.
418. class SettingsForm : public QDialog
419. {
420. Q\_OBJECT
422. public:
423. explicit SettingsForm(QWidget \*parent = nullptr);
424. ~SettingsForm();
426. double p, p1, p2, p3;
427. double old\_p, old\_p1, old\_p2, old\_p3;
428. double usd, eur, old\_usd, old\_eur;
429. bool is\_saved = false;
431. void check\_percents();
433. private slots:
434. void on\_spin\_p\_1\_valueChanged(double arg1);
436. void on\_spin\_p\_2\_valueChanged(double arg1);
438. void on\_spin\_p\_3\_valueChanged(double arg1);
440. void on\_spin\_p\_4\_valueChanged(double arg1);
442. void on\_button\_discharge\_clicked();
444. void on\_button\_save\_clicked();
446. void on\_spin\_usd\_valueChanged(double arg1);
448. void on\_spin\_eur\_valueChanged(double arg1);
450. private:
451. Ui::SettingsForm \*ui;
452. };

5.1. Метод check\_percents используется для передачи на поля актуальных значений процентных ставок и курсов валют при открытии окна.

1. void SettingsForm::check\_percents()
2. {
3. ui->spin\_p\_1->setValue(p);
4. ui->spin\_p\_2->setValue(p1);
5. ui->spin\_p\_3->setValue(p2);
6. ui->spin\_p\_4->setValue(p3);
7. ui->spin\_usd->setValue(usd);
8. ui->spin\_eur->setValue(eur);
9. }

5.2. Слоты on\_spin\_p\_1\_valueChanged, on\_spin\_p\_2\_valueChanged, on\_spin\_p\_3\_valueChanged, on\_spin\_p\_4\_valueChanged при изменении значений в полях ввода процентных ставок сохраняют измененные значения в соответствующие переменные.

1. void SettingsForm::on\_spin\_p\_1\_valueChanged(double arg1)
2. {
3. p = ui->spin\_p\_1->value();
4. }
6. void SettingsForm::on\_spin\_p\_2\_valueChanged(double arg1)
7. {
8. p1 = ui->spin\_p\_2->value();
9. }
11. void SettingsForm::on\_spin\_p\_3\_valueChanged(double arg1)
12. {
13. p2 = ui->spin\_p\_3->value();
14. }
16. void SettingsForm::on\_spin\_p\_4\_valueChanged(double arg1)
17. {
18. p3 = ui->spin\_p\_4->value();
19. }

5.3. Слоты on\_spin\_usd\_valueChanged, on\_spin\_eur\_valueChanged при изменении значений в полях ввода курсов валют сохраняют измененные значения в соответствующие переменные.

1. void SettingsForm::on\_spin\_usd\_valueChanged(double arg1)
2. {
3. usd = ui->spin\_usd->value();
4. }
6. void SettingsForm::on\_spin\_eur\_valueChanged(double arg1)
7. {
8. eur = ui->spin\_eur->value();
9. }

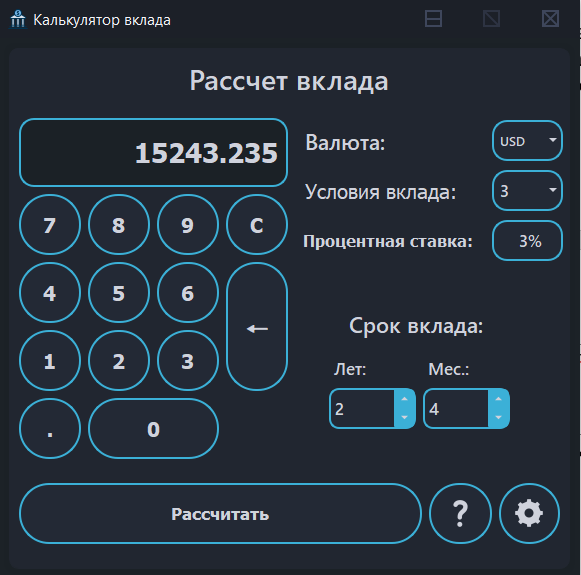
5.4. Слот on\_button\_discharge\_clicked по нажатии на кнопку сброса сбрасывает значения процентных ставок и курсов валют до последних сохраненных.

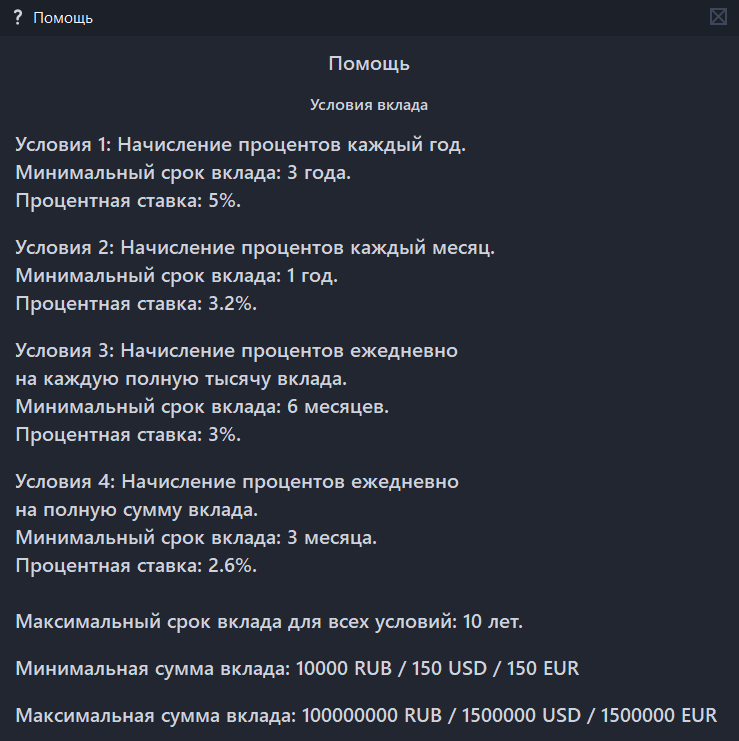
1. void SettingsForm::on\_button\_discharge\_clicked()
2. {
3. p = old\_p; ui->spin\_p\_1->setValue(p);
4. p1 = old\_p1; ui->spin\_p\_2->setValue(p1);
5. p2 = old\_p2; ui->spin\_p\_3->setValue(p2);
6. p3 = old\_p3; ui->spin\_p\_4->setValue(p3);
7. usd = old\_usd; ui->spin\_usd->setValue(usd);
8. eur = old\_eur; ui->spin\_eur->setValue(eur);
9. }

5.5. Слот on\_button\_save\_clicked по нажатии на кнопку сохранения меняет значение is\_saved и закрывает окно. Затем значения переменных сохраняются в классе MainWindow.

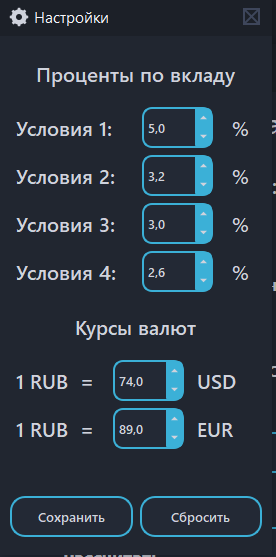
1. void SettingsForm::on\_button\_save\_clicked()
2. {
3. is\_saved = true;
4. close();
5. }
6. **Демонстрация работы калькулятора.**

6.1. Основное окно.

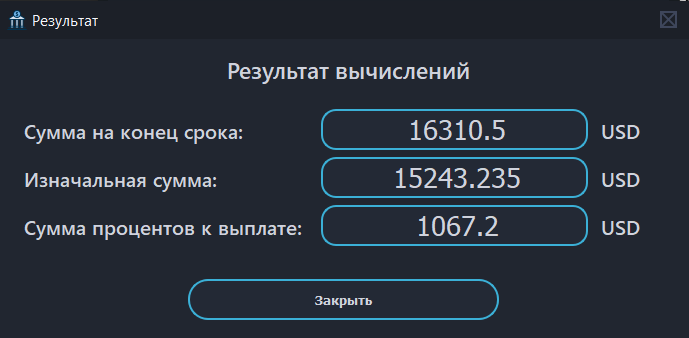


6.2. Окно помощи.

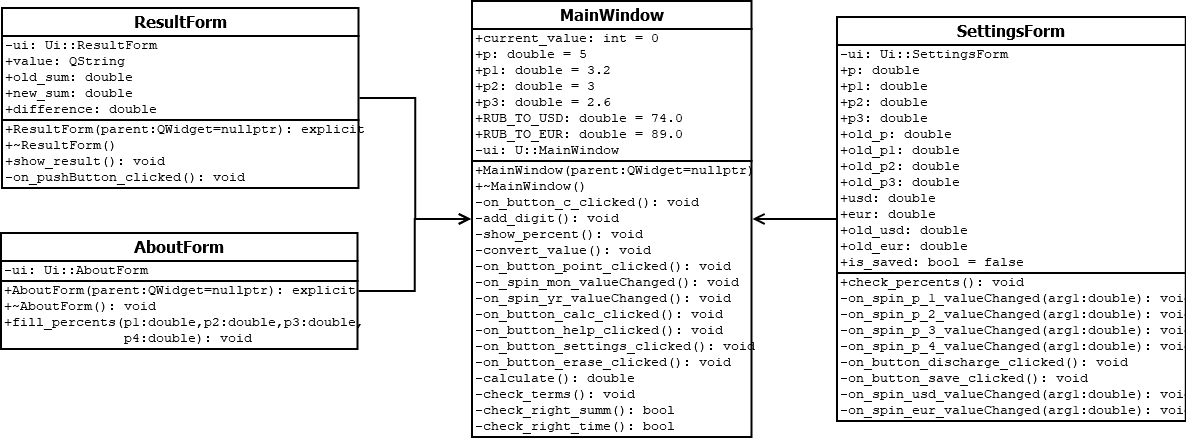
6.3. Окно настроек.



6.4. Вывод результата.

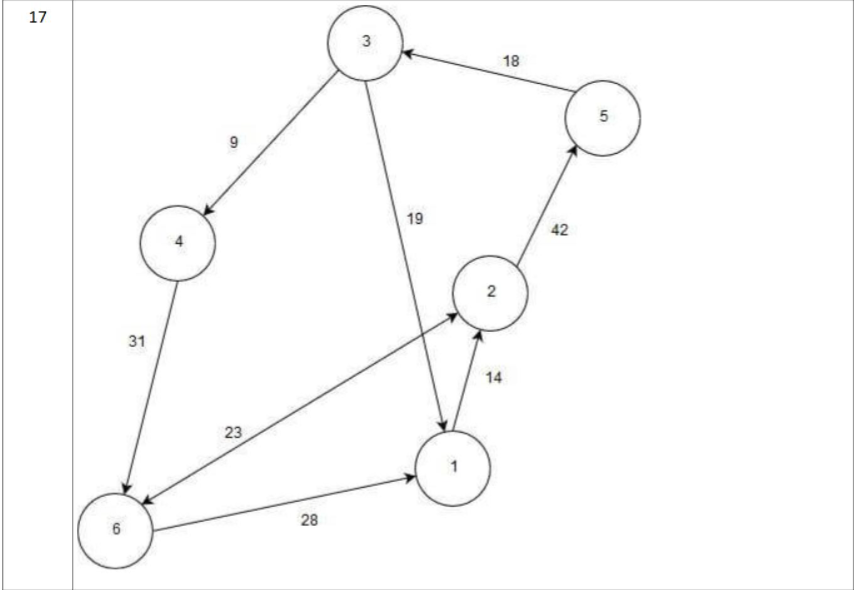


**UML-диаграмма**



**Задача коммивояжера**

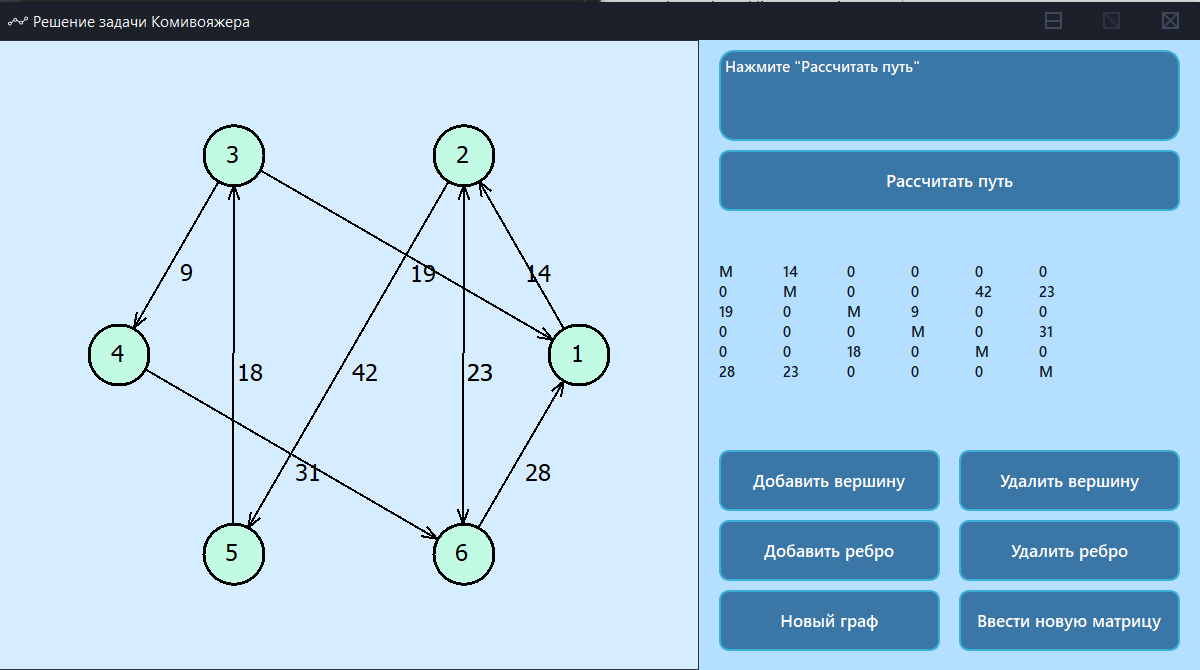
**Постановка задачи**

****

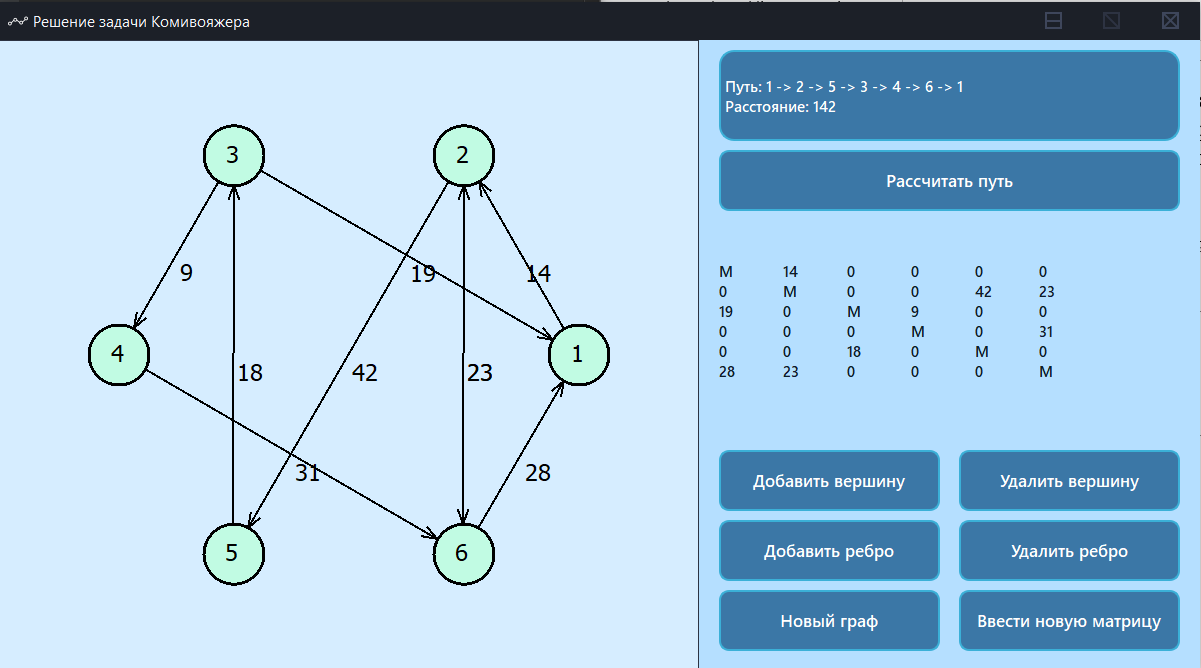
**Анализ задачи**

1. Описание алгоритма работы программы.
   1. Ввод данных осуществляется в поля, которые присутствуют в формах DelEdgeForm, DelVertForm, NewEdgeForm, NewMatrix\_6, NewMatrixForm.
   2. Отрисовка. Вершины графа располагаются по окружности. Между вершинами рисуются ребра графа со стрелками.
   3. Решение задачи коммивояжера выводится а поле на главном окне после нажатия кнопки “Рассчитать путь”.
2. -7. Описание структуры программы.
   * 1. Класс Graph наследуется от класса QMainWindow. Класс содержит матрицу смежности графа и целочисленную переменную, хранящую количество вершин графа и необходимые методы и слоты.
3. class Graph : public QMainWindow
4. {
5. Q\_OBJECT
7. private:
8. Ui::Graph \*ui;
9. QGraphicsScene \*scene;
11. public:
12. int kol;
13. std::vector<std::vector<int>> matrix;
15. Graph(QWidget \*parent = nullptr);
16. ~Graph();
17. int GetWeight(const int vert1, const int vert2);
18. void DrawLines(QGraphicsScene\* scene);
19. void DrawArrows(QGraphicsScene\* scene);
20. void DrawNodes(QGraphicsScene\* scene);
21. void DrawGraph(QGraphicsScene\* scene);
22. void Draw();
23. void Add\_Vert();
24. void Del\_Vert(int vert);
25. void New\_Edge();
26. void Add\_Edge(int source, int target, int weight);
27. void Del\_Edge(int src, int trg);
28. void New\_Matrix\_6();
30. QString komi();
31. QString Print\_Matrix();
32. private slots:
33. void on\_b\_addVert\_clicked();
34. void on\_b\_komi\_clicked();
35. void on\_b\_newMatrix\_clicked();
36. void on\_b\_addEdge\_clicked();
37. void on\_b\_delVert\_clicked();
38. void on\_b\_delEdge\_clicked();
39. void on\_b\_newMatrix\_6\_clicked();
40. };
    1. Метод komi для решения задачи коммивояжера методом ветвей и границ.
41. QString Graph::komi() //решение задачи комивояжера
42. {
43. int n;
44. int\* res;
45. int\*\* help;
46. int\*\*\* mat;
48. n = kol;
49. help = new int\* [n];
50. res = new int[n];
51. mat = new int\*\* [n];
53. for (int i = 0; i <= n; i++) {
54. help[i] = new int[n];
55. }
57. for (int i = 0; i < n; i++) //замена нулей матрицы на нулевые указатели
58. {
59. mat[i] = new int\* [n];
60. for (int j = 0; j < n; j++) {
61. if (matrix[i][j] == 0) {
62. mat[i][j] = nullptr;
63. continue;
64. }
65. mat[i][j] = new int(matrix[i][j]);
66. }
67. }
69. for (int l = 0; l < n; l++)
70. {
71. for (int i = 0; i < n; i++) //поиск минимальных элементов и редукция матрицы
72. {
73. int min = 1000000;
74. for (int j = 0; j < n; j++)
75. if (mat[i][j] && min > \* mat[i][j])
76. min = \*mat[i][j];
77. for (int j = 0; j < n; j++)
78. if (mat[i][j])
79. \*mat[i][j] -= min;
80. }
81. for (int j = 0; j < n; j++)
82. {
83. int min = 1000000;
84. for (int i = 0; i < n; i++)
85. if (mat[i][j] && min > \* mat[i][j])
86. min = \*mat[i][j];
87. for (int i = 0; i < n; i++)
88. if (mat[i][j])
89. \*mat[i][j] -= min;
90. }
92. for (int i = 0; i < n; i++)
93. for (int j = 0; j < n; j++)
94. help[i][j] = 0;
96. for (int i = 0; i < n; i++) //оценка нулей
97. for (int j = 0; j < n; j++)
98. {
99. if (mat[i][j] && !\*mat[i][j])
100. {
101. int hmin = 1000000;
102. int vmin = 1000000;
104. for (int l = 0; l < n; l++)
105. if (l != i && mat[l][j] && hmin > \* mat[l][j]) hmin = \*mat[l][j];
106. for (int l = 0; l < n; l++)
107. if (l != j && mat[i][l] && vmin > \* mat[i][l]) vmin = \*mat[i][l];
108. help[i][j] = hmin + vmin;
109. }
110. }
112. int mcost = 0, mi = 0, mj = 0; //чистка карты
113. for (int i = 0; i < n; i++)
114. for (int j = 0; j < n; j++)
115. if (mat[i][j] && mcost < help[i][j]) {
116. mcost = help[i][j];
117. mi = i;
118. mj = j;
119. }
120. res[mi] = mj;
122. for (int i = 0; i < n; i++)
123. mat[i][mj] = nullptr;
125. for (int i = 0; i < n; i++)
126. mat[mi][i] = nullptr;
128. mat[mj][mi] = nullptr;
129. }
130. int s = 0; //подсчет длины пути
131. for (int i = 0, j = 0; i < n; i++)
132. {
133. j = res[i];
134. s += matrix[i][j];
135. }
136. QString tmp = ""; //вывод кратчайшего маршрута и его длинны
137. tmp += QString("\nПуть: ");
138. int temp = 0;
139. for (int l = 0; l < n;) {
140. for (int i = 0, j = 0; i < n; i++) {
141. if (temp == 0 || i + 1 == temp) {
142. if (temp == 0) tmp += QString::number(i + 1);
143. j = res[i];
144. temp = j + 1;
145. if (temp > 0) tmp += QString(" -> ") + QString::number(temp);
146. l++;
147. }
148. }
149. }
150. tmp += QString("\nРасстояние: ") + QString::number(s);
151. return tmp;
152. }
     1. В конструкторе класса заранее введена матрица смежности из варианта 17.
153. Graph::Graph(QWidget \*parent)
154. : QMainWindow(parent)
155. , ui(new Ui::Graph)
156. {
157. ui->setupUi(this);
158. kol = 6;
159. matrix = { {0, 14, 0, 0, 0, 0 },
160. {0, 0, 0, 0, 42, 23 },
161. {19, 0, 0, 9, 0, 0 },
162. {0, 0, 0, 0, 0, 31 },
163. {0, 0, 18, 0, 0, 0 },
164. {28, 23, 0, 0, 0, 0 } };
165. Draw();
166. }
     1. Метод Print\_Matrix для вывода матрицы смежности.
167. QString Graph::Print\_Matrix() //вывод матрицы смежности
168. {
169. QString tmp = "";
170. for (int i = 0; i < matrix.size(); i++)
171. {
172. for (int j = 0; j < matrix.size(); j++)
173. {
174. if (i == j)
175. tmp += "M\t";
176. else
177. tmp += QString::number(matrix[i][j]) + "\t";
178. }
179. tmp += "\n";
180. }
181. return tmp;
182. }
     1. Метод GetWeight для получения веса ребра между двумя вершинами.
183. int Graph::GetWeight(const int vert1, const int vert2) //получение веса ребра
184. {
185. if (vert1 >= 0 && vert1 < matrix.size() && vert2 >=0 && vert2 < matrix.size())
186. return matrix[vert1][vert2];
187. else
188. return 0;
189. }
     1. Метод DrawLines для вычисления координат и отрисовки ребер графа.
190. void Graph::DrawLines(QGraphicsScene\* scene) //отрисовка ребер графа
191. {
192. QPen pen(Qt::black);
193. pen.setWidth(2);
194. int centerX= scene->width() / 2,
195. centerY = scene->height() / 2;
196. double iter = 2 \* 3.1415 / matrix.size();
197. for (int i = 0; i < matrix.size(); i++){
198. for (int j = 0; j < matrix.size(); j++){
199. if (GetWeight(i, j) > 0){
200. int vert\_1\_x = centerX + cos(iter \* i) \* radiusB,
201. vert\_1\_y = centerY - sin(iter \* i) \* radiusB,
202. vert\_2\_x = centerX + cos(iter \* j) \* radiusB,
203. vert\_2\_y = centerY - sin(iter \* j) \* radiusB;
204. scene->addLine(vert\_1\_x, vert\_1\_y, vert\_2\_x, vert\_2\_y, pen);
205. QGraphicsTextItem \*number = scene->addText(QString::number(GetWeight(i, j)), QFont("Times", 14));
206. number->setPos((vert\_2\_x + vert\_1\_x) / 2, (vert\_2\_y + vert\_1\_y) / 2);
207. }
208. }
209. }
210. }
     1. Метод DrawArrows для вычисления координат и отрисовки стрелок ребер графа.
211. void Graph::DrawArrows(QGraphicsScene\* scene) //отрисовка стрелок
212. {
213. QPen pen(Qt::black);
214. pen.setWidth(2);
215. int centerX= scene->width() / 2,
216. centerY = scene->height() / 2;
217. double len = 15.0;
218. double ostr = 0.35;
219. double arr\_x, arr\_y;
220. double iter = 2 \* 3.1415 / matrix.size();
221. for (int i = 0; i < matrix.size(); i++){
222. for (int j = 0; j < matrix.size(); j++){
223. if (GetWeight(i, j) > 0){
224. int vert\_1\_x = centerX + cos(iter \* i) \* radiusB,
225. vert\_1\_y = centerY - sin(iter \* i) \* radiusB,
226. vert\_2\_x = centerX + cos(iter \* j) \* radiusB,
227. vert\_2\_y = centerY - sin(iter \* j) \* radiusB;
228. double angle = atan2(((double)vert\_2\_y - vert\_1\_y), ((double)(vert\_2\_x - vert\_1\_x)));
229. //double t = radiusA / sqrt((vert\_2\_x - vert\_1\_x) \* (vert\_2\_x - vert\_1\_x) + (vert\_2\_y - vert\_1\_y) \* (vert\_2\_y - vert\_1\_y));
230. arr\_x = (vert\_2\_x - (radiusA \* cos(angle))) - (len \* cos(angle + ostr));
231. arr\_y = (vert\_2\_y - (radiusA \* sin(angle))) - (len \* sin(angle + ostr));
232. scene->addLine(vert\_2\_x - (radiusA \* cos(angle)), vert\_2\_y - (radiusA \* sin(angle)), arr\_x, arr\_y, pen);
233. arr\_x = (vert\_2\_x - (radiusA \* cos(angle))) - (len \* cos(angle - ostr));
234. arr\_y = (vert\_2\_y - (radiusA \* sin(angle))) - (len \* sin(angle - ostr));
235. scene->addLine(vert\_2\_x - (radiusA \* cos(angle)), vert\_2\_y - (radiusA \* sin(angle)), arr\_x, arr\_y, pen);
236. }
237. }
238. }
239. }
     1. Метод DrawNodes для вычисления координат и отрисовки вершин графа.
240. void Graph::DrawNodes(QGraphicsScene\* scene) //отрисовка вершин графа
241. {
242. QPen pen(Qt::black);
243. pen.setWidth(3);
244. int centerX= scene->width() / 2,
245. centerY = scene->height() / 2;
246. double iter = 2 \* 3.1415 / matrix.size();
247. for (int i = 0; i < matrix.size(); i++){
248. scene->addEllipse(centerX + cos(iter \* i) \* radiusB - radiusA,
249. centerY - sin(iter \* i) \* radiusB - radiusA, 2\*radiusA, 2\*radiusA,
250. pen, QBrush(QColor(193,251,227)));
252. QGraphicsTextItem \*number = scene->addText(QString::number(i+1), QFont("Times", 14));
253. number->setPos(centerX + cos(iter \* i) \* radiusB - radiusA + 18,
254. centerY - sin(iter \* i) \* radiusB - radiusA + 10);
255. }
256. }
     1. Метод DrawGraph, который вызывает методы DrawLines, DrawArrows и DrawNodes.
257. void Graph::DrawGraph(QGraphicsScene\* scene) //отрисовка графа
258. {
259. DrawLines(scene);
260. DrawArrows(scene);
261. DrawNodes(scene);
262. }
     1. Метод Draw для отрисовки графической сцены с графом.
263. void Graph::Draw() //отрисовка сцены с графом
264. {
265. scene = new QGraphicsScene();
266. scene->setSceneRect(0, 0, 700, 630);
268. ui->graphicsView->setScene(scene);
269. scene->clear();
270. scene->setBackgroundBrush(QBrush(QColor(214,237,255), Qt::SolidPattern));
271. ui->label\_Matrix->setText(Print\_Matrix());
272. DrawGraph(scene);
273. }
     1. Метод Add\_Vert для добавления вершины в граф.
274. void Graph::Add\_Vert() //добавление вершины
275. {
276. for (int i = 0; i < kol; i++)
277. {
278. matrix[i].push\_back(0);
279. }
281. matrix.push\_back({});
282. kol++;
283. for (int i = 0; i < kol; i++)
284. {
285. matrix[kol - 1].push\_back(0);
286. }
288. }
     1. Метод Del\_Vert для удаления вершины из графа.
289. void Graph::Del\_Vert(int vert) //удаление вершины
290. {
291. matrix.erase(matrix.begin() + vert);
292. kol--;
293. for (int i = 0; i < kol; i++)
294. {
295. matrix[i].erase(matrix[i].begin() + vert);
296. }
297. }
     1. Метод Del\_Edge для удаления ребра из графа.
298. void Graph::Del\_Edge(int src, int trg) //удаление ребра
299. {
300. matrix[src - 1][trg - 1] = 0;
301. }
     1. Метод Add\_Edge для добавления ребра в граф.
302. void Graph::Add\_Edge(int source, int target, int weight) //добавление ребра
303. {
304. if (source > 0 && source <= kol && target > 0 && target <= kol && weight > 0)
305. {
306. matrix[source - 1][target - 1] = weight;
307. }
308. }
     1. Метод New\_Matrix\_6 для создания нового графа путем ввода новой матрицы смежности.
309. void Graph::New\_Matrix\_6() //ввод новой матрицы
310. {
311. NewMatrix\_6 window;
312. window.setWindowFlags (window.windowFlags() & ~Qt::WindowContextHelpButtonHint);
313. window.setModal(true);
314. window.exec();
315. if (window.isSaved)
316. {
317. matrix.clear();
318. kol = 6;
319. for (int i = 0; i < kol; i++)
320. {
321. matrix.push\_back({});
322. for (int j = 0; j < kol; j++)
323. {
324. matrix[i].push\_back(0);
325. }
326. }
328. matrix[0][0] = window.s\_0\_0;
329. matrix[0][1] = window.s\_0\_1;
330. matrix[0][2] = window.s\_0\_2;
331. matrix[0][3] = window.s\_0\_3;
332. matrix[0][4] = window.s\_0\_4;
333. matrix[0][5] = window.s\_0\_5;
335. matrix[1][0] = window.s\_1\_0;
336. matrix[1][1] = window.s\_1\_1;
337. matrix[1][2] = window.s\_1\_2;
338. matrix[1][3] = window.s\_1\_3;
339. matrix[1][4] = window.s\_1\_4;
340. matrix[1][5] = window.s\_1\_5;
342. matrix[2][0] = window.s\_2\_0;
343. matrix[2][1] = window.s\_2\_1;
344. matrix[2][2] = window.s\_2\_2;
345. matrix[2][3] = window.s\_2\_3;
346. matrix[2][4] = window.s\_2\_4;
347. matrix[2][5] = window.s\_2\_5;
349. matrix[3][0] = window.s\_3\_0;
350. matrix[3][1] = window.s\_3\_1;
351. matrix[3][2] = window.s\_3\_2;
352. matrix[3][3] = window.s\_3\_3;
353. matrix[3][4] = window.s\_3\_4;
354. matrix[3][5] = window.s\_3\_5;
356. matrix[4][0] = window.s\_4\_0;
357. matrix[4][1] = window.s\_4\_1;
358. matrix[4][2] = window.s\_4\_2;
359. matrix[4][3] = window.s\_4\_3;
360. matrix[4][4] = window.s\_4\_4;
361. matrix[4][5] = window.s\_4\_5;
363. matrix[5][0] = window.s\_5\_0;
364. matrix[5][1] = window.s\_5\_1;
365. matrix[5][2] = window.s\_5\_2;
366. matrix[5][3] = window.s\_5\_3;
367. matrix[5][4] = window.s\_5\_4;
368. matrix[5][5] = window.s\_5\_5;
369. }
371. }
     1. Метод New\_Edge используется для вывода окна для добавления ребра, ввода в него данных и последующего добавления ребра в граф.
372. void Graph::New\_Edge() //вывод окна для добавления ребра
373. {
374. NewEdgeForm window;
375. window.setWindowFlags (window.windowFlags() & ~Qt::WindowContextHelpButtonHint);
376. window.setModal(true);
377. window.max\_kol = kol + 1;
378. window.check\_max();
379. window.exec();
380. if(window.isSaved)
381. {
382. int source = window.source;
383. int target = window.target;
384. int weight = window.weight;
385. bool way = window.way;
386. Add\_Edge(source, target, weight);
387. if (way)
388. {
389. Add\_Edge(target, source, weight);
390. }
391. }
392. }
     1. Слот on\_b\_addVert\_clicked по нажатию на кнопку вызывает метод Add\_Vert, затем заново отрисовывает граф с помощью функции Draw.
393. void Graph::on\_b\_addVert\_clicked()
394. {
395. Add\_Vert();
396. Draw();
397. }
     1. Слот on\_b\_komi\_clicked по нажатию на кнопку выводит на QLabel результат решения задачи коммивояжера для данного графа.
398. void Graph::on\_b\_komi\_clicked()
399. {
400. ui->label\_komi->setText(komi());
401. }
     1. Слот on\_b\_newMatrix\_clicked по нажатию на кнопку вызывает новое окно для ввода новой матрицы.
402. void Graph::on\_b\_newMatrix\_clicked()
403. {
404. NewMatrixForm new\_mat;
405. new\_mat.setWindowFlags (new\_mat.windowFlags() & ~Qt::WindowContextHelpButtonHint);
406. new\_mat.setModal(true);
407. new\_mat.exec();
408. if (new\_mat.isSaved)
409. {
410. matrix.clear();
411. kol = 0;
412. for (int i = 0; i < new\_mat.verts; i++)
413. {
414. Add\_Vert();
415. }
416. int verts\_num = new\_mat.edges;
417. for (int i = 0; i < verts\_num; i++)
418. {
419. New\_Edge();
420. }
421. }
422. Draw();
423. }
424. 1. Слот on\_b\_addEdge\_clicked по нажатию на кнопку вызывает методы New\_Edge и Draw.
425. void Graph::on\_b\_addEdge\_clicked()
426. {
427. New\_Edge();
428. Draw();
429. }
     1. Слот on\_b\_delVert\_clicked по нажатию на кнопку вызывает окно для удаления вершины.
430. void Graph::on\_b\_delVert\_clicked()
431. {
432. DelVertForm window;
433. window.setWindowFlags (window.windowFlags() & ~Qt::WindowContextHelpButtonHint);
434. window.setModal(true);
435. window.max\_kol = kol + 1;
436. window.check\_max();
437. window.exec();
438. if (window.isSaved)
439. {
440. Del\_Vert(window.kol - 1);
441. }
442. Draw();
443. }
     1. Слот on\_b\_delEdge\_clicked по нажатию на кнопку вызывает окно для удаления ребра.
444. void Graph::on\_b\_delEdge\_clicked()
445. {
446. DelEdgeForm window;
447. window.setWindowFlags (window.windowFlags() & ~Qt::WindowContextHelpButtonHint);
448. window.setModal(true);
449. window.max\_kol = kol + 1;
450. window.check\_max();
451. window.exec();
452. if (window.isSaved)
453. {
454. int source = window.source;
455. int target = window.target;
456. Del\_Edge(source, target);
457. Draw();
458. }
459. }
     1. Слот on\_b\_newMatrix\_6\_clicked по нажатию на кнопку вызывает метод New\_Matrix\_6 для ввода новой матрицы смежности.
460. void Graph::on\_b\_newMatrix\_6\_clicked()
461. {
462. New\_Matrix\_6();
463. Draw();
464. }
465. Класс DelEdgeForm наследуется от класса QDialog. Класс имеет целочисленные переменные для хранения количества вершин в графе, начальной вершины ребра и конечной вершины ребра, булевую переменную isSaved, а также необходимые методы и слоты.
466. class DelEdgeForm : public QDialog
467. {
468. Q\_OBJECT
470. public:
471. explicit DelEdgeForm(QWidget \*parent = nullptr);
472. ~DelEdgeForm();
473. int max\_kol;
474. int source, target;
475. bool isSaved = false;
477. void check\_max();
479. private slots:
480. void on\_button\_clicked();
482. private:
483. Ui::DelEdgeForm \*ui;
484. };
     1. Метод check\_max для установки полям ввода максимального значения, равного количеству вершин в графе, во избежание некорректного ввода.
485. void DelEdgeForm::check\_max()
486. {
487. ui->spin\_source->setMaximum(max\_kol);
488. ui->spin\_target->setMaximum(max\_kol);
489. }
     1. Слот on\_button\_clicked по нажатию кнопки сохраняет введенные данные в переменные и их значения передаются в главное окно.
490. void DelEdgeForm::on\_button\_clicked()
491. {
492. isSaved = true;
493. source = ui->spin\_source->value();
494. target = ui->spin\_target->value();
495. close();
496. }
497. Класс DelVertForm наследуется от класса QDialog. Класс имеет целочисленные переменные для хранения количества вершин в графе и введенной вершины для удаления, булевую переменную isSaved, а также необходимые методы и слоты.
     1. Метод check\_max для установки полям ввода максимального значения, равного количеству вершин в графе, во избежание некорректного ввода.
498. void DelVertForm::check\_max()
499. {
500. ui->spin\_vert->setMaximum(max\_kol);
501. }
     1. Слот on\_button\_clicked по нажатию кнопки сохраняет введенные данные в переменные и их значения передаются в главное окно.
502. void DelVertForm::on\_button\_clicked()
503. {
504. isSaved = true;
505. kol = ui->spin\_vert->value();
506. close();
507. }
508. Класс NewEdgeForm наследуется от класса QDialog. Класс имеет целочисленные переменные для хранения количества вершин в графе, начальной вершины ребра, конечной вершины ребра, веса ребра, булевой переменных way и isSaved, а также необходимые методы и слоты.
     1. Метод check\_max для установки полям ввода максимального значения, равного количеству вершин в графе, во избежание некорректного ввода.
509. void NewEdgeForm::check\_max()
510. {
511. ui->spin\_source->setMaximum(max\_kol);
512. ui->spin\_target->setMaximum(max\_kol);
513. }
     1. Слот on\_button\_clicked по нажатию кнопки сохраняет введенные данные в переменные и их значения передаются в главное окно.
514. void NewEdgeForm::on\_button\_clicked()
515. {
516. isSaved = true;
517. source = ui->spin\_source->value();
518. target = ui->spin\_target->value();
519. weight = ui->spin\_weight->value();
520. way = ui->checkWay->checkState();
521. close();
522. }
523. Класс NewMatrix\_6 наследуется от класса QDialog. Класс имеет целочисленные переменные для хранения новой матрицы смежности, булевую переменную isSaved и слот on\_pushButton\_clicked.
     1. Слот on\_pushButton\_clicked по нажатию кнопки сохраняет введенные данные в переменные и их значения передаются в главное окно.
524. void NewMatrix\_6::on\_pushButton\_clicked()
525. {
526. isSaved = true;
527. s\_0\_0 = 0; s\_0\_1 = ui->s\_0\_1->value(); s\_0\_2 = ui->s\_0\_2->value();
528. s\_0\_3 = ui->s\_0\_3->value(); s\_0\_4 = ui->s\_0\_4->value(); s\_0\_5 = ui->s\_0\_5->value();
530. s\_1\_0 = ui->s\_1\_0->value(); s\_1\_1 = 0; s\_1\_2 = ui->s\_1\_2->value();
531. s\_1\_3 = ui->s\_1\_3->value(); s\_1\_4 = ui->s\_1\_4->value(); s\_1\_5 = ui->s\_1\_5->value();
533. s\_2\_0 = ui->s\_2\_0->value(); s\_2\_1 = ui->s\_2\_1->value(); s\_2\_2 = 0;
534. s\_2\_3 = ui->s\_2\_3->value(); s\_2\_4 = ui->s\_2\_4->value(); s\_2\_5 = ui->s\_2\_5->value();
536. s\_3\_0 = ui->s\_3\_0->value(); s\_3\_1 = ui->s\_3\_1->value(); s\_3\_2 = ui->s\_3\_2->value();
537. s\_3\_3 = 0; s\_3\_4 = ui->s\_3\_4->value(); s\_3\_5 = ui->s\_3\_5->value();
539. s\_4\_0 = ui->s\_4\_0->value(); s\_4\_1 = ui->s\_4\_1->value(); s\_4\_2 = ui->s\_4\_2->value();
540. s\_4\_3 = ui->s\_4\_3->value(); s\_4\_4 = 0; s\_4\_5 = ui->s\_4\_5->value();
542. s\_5\_0 = ui->s\_5\_0->value(); s\_5\_1 = ui->s\_5\_1->value(); s\_5\_2 = ui->s\_5\_2->value();
543. s\_5\_3 = ui->s\_5\_3->value(); s\_5\_4 = ui->s\_5\_4->value(); s\_5\_5 = 0;
545. close();
546. }
547. Класс NewMatrixForm наследуется от класса QDialog. Класс имеет целочисленные переменные для хранения количества вершин и ребер нового графа, булевую переменную isSaved, а также слот on\_button\_clicked.
     1. Слот on\_button\_clicked по нажатию кнопки сохраняет введенные данные в переменные и их значения передаются в главное окно.
548. void NewMatrixForm::on\_button\_clicked()
549. {
550. verts = ui->spinVerts->value();
551. edges = ui->spinEdges->value();
552. isSaved = true;
553. close();
554. }
555. **Демонстрация работы программы.**

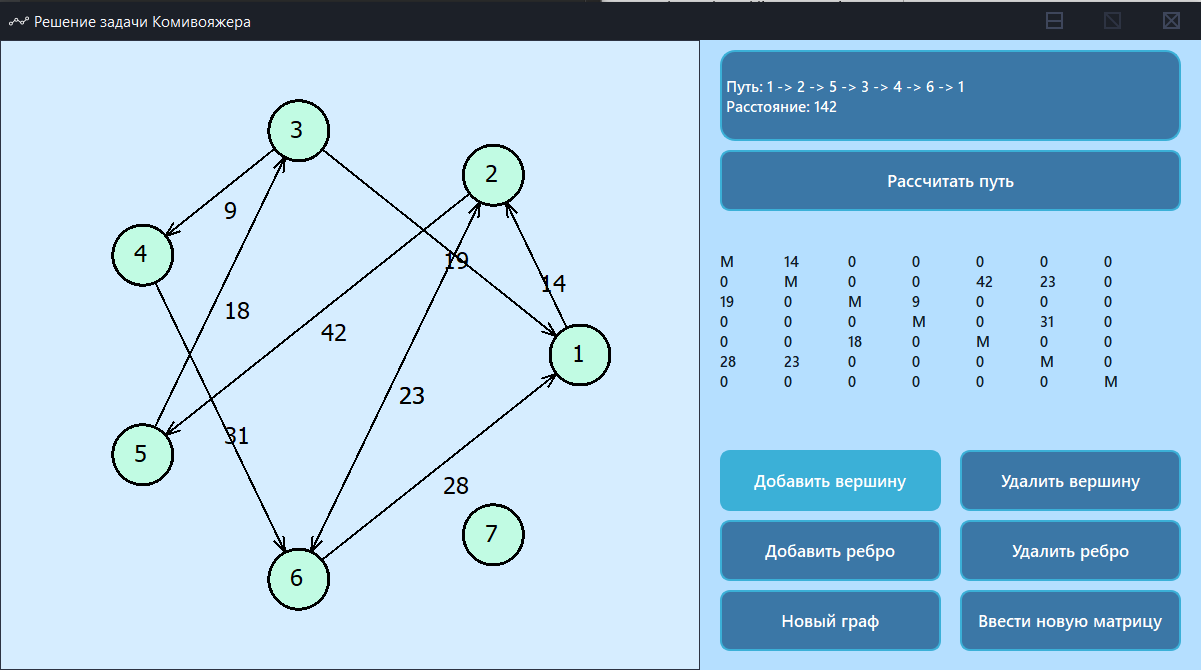
8.1. Главное окно



8.2. Решение задачи для данного графа.



8.3. Добавление вершины в граф.



8.4. Окна удаления вершины, добавления ребра, удаления ребра, создания графа и ввода новой матрицы.

