**C++ 语 言 程 序 设 计**

实

验

报

告

实 验 三

姓名： 葛旭

学号： 190320517

班级： 自动化五班

**一 实验项目**

* 1. 编写矩阵类Matrix\_4x4，实现矩阵初始化、求逆、转置、访问等基本功能
  2. 基于运算符重载，实现矩阵的加减乘、输入输出的操作
  3. 具体内容如下：

**class Matrix\_4x4**

**{**

**private:**

**double matrix[4][4];**

**public:**

**//默认构造函数，初始化矩阵为单位阵**

**//带参数构造函数，用一个4x4的二维数组初始化**

**//拷贝构造函数**

**//重载 = 运算符，参数为矩阵对象**

**//重载 = 运算符，参数为一个4x4的二维数组**

**// 重载算术运算符 + - \***

**// 重载 ^ 运算符为矩阵的i次幂（如果i为负数，如何处理？）**

**// 重载 [ ] 运算符，实现双下标方式访问矩阵元素（该功能已经实现，无需自己写）**

**const double \* operator[] (const int i) const {return matrix[i];}**

**double \* operator[] (const int i) {return matrix[i];}**

**// 重载输入<< 和输出 >>**

**Matrix\_4x4 inverse(); //矩阵求逆，不改变当前矩阵值，返回逆矩阵**

**Matrix\_4x4 transpose(); //矩阵转置，不改变当前矩阵值，返回转置矩阵**

**};**

**二 实验原理**

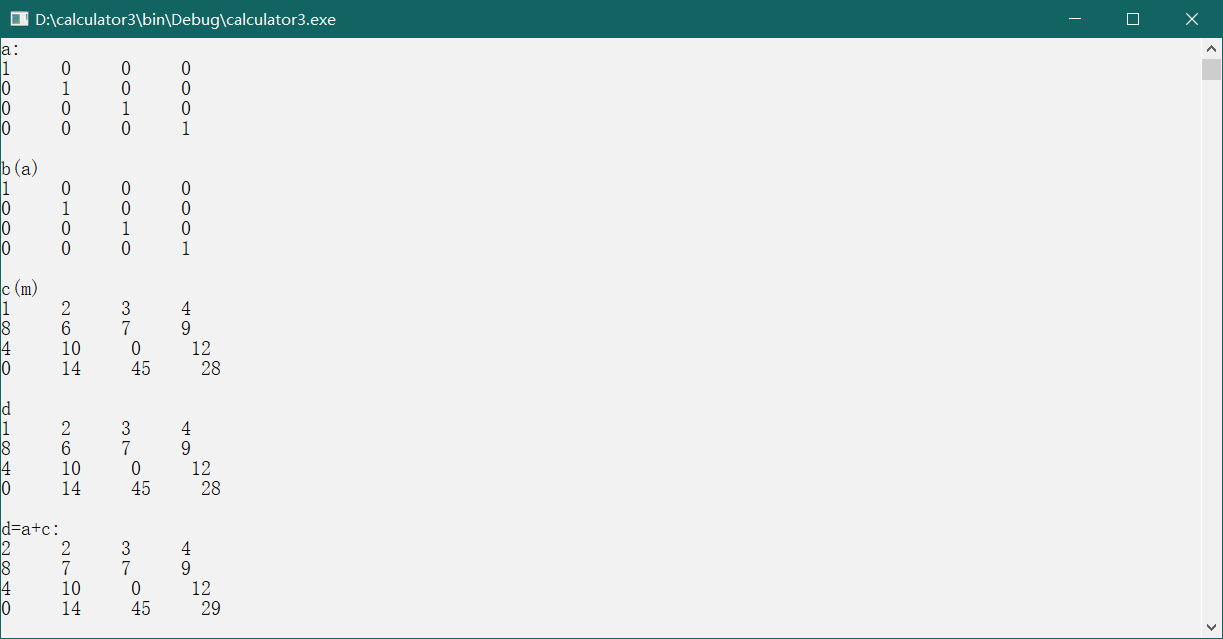
1. 说明矩阵求逆与行列式的原理与操作方法（给出算法的流程图与简要说明，可参考附件PPT内容）

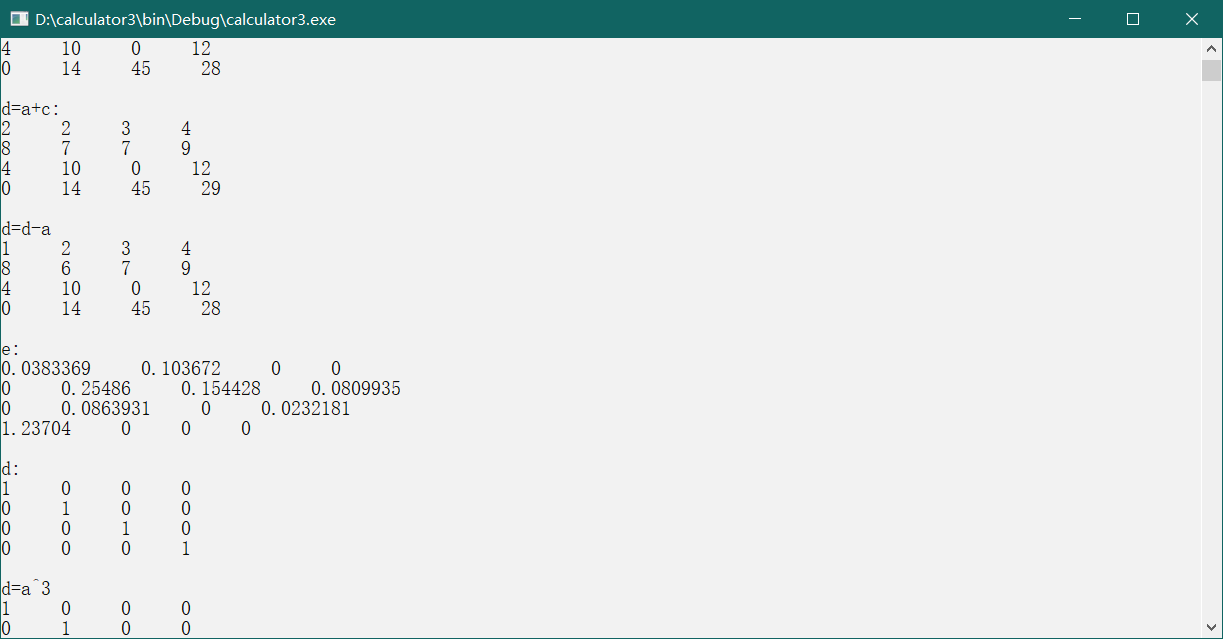




其中利用了高斯-若尔法将矩阵经过有限次初等变换变为单位阵。

1. 矩阵类程序实现与结果（给出算法的源代码，说明关键代码的操作含义，给出运行结果）
2. #include <iostream>
3. #include <string>
4. #include <math.h>
6. **using** **namespace** std;
8. **class** Matrix\_4x4
9. {
10. **private**:
11. **double** matrix[4][4];
12. **public**:
13. //默认构造函数
14. Matrix\_4x4()
15. {
16. **for** (**int** i = 0; i < 4; i++)
17. {
18. **for** (**int** j = 0; j < 4; j++)
19. {
20. matrix[i][j] = (i == j) ? 1 : 0;
21. }
22. }
23. }
24. //带参构造函数
25. Matrix\_4x4(**const** **double** m[4][4])
26. {
27. **for**(**int** i=0;i<4;i++)
28. {
29. **for**(**int** j=0;j<4;j++)
30. {
31. matrix[i][j]=m[i][j];
32. }
33. }
34. }
35. //拷贝构造函数
36. Matrix\_4x4(Matrix\_4x4 **const** &m)
37. {
38. **for**(**int** i=0;i<4;i++)
39. {
40. **for**(**int** j=0;j<4;j++)
41. {
42. matrix[i][j]=m[i][j];
43. }
44. }
45. }
46. //重载 = 运算符，参数为矩阵对象
47. Matrix\_4x4 **const** & operator = (Matrix\_4x4 **const** &m);
48. //重载 = 运算符，参数为一个4x4的二维数组
49. Matrix\_4x4 **const** & operator = (**const** **double** m[4][4]);
50. //重载算术运算符 + - \*
51. **friend** Matrix\_4x4 operator + (Matrix\_4x4 **const** &m1,Matrix\_4x4 **const** &m2);
52. **friend** Matrix\_4x4 operator - (Matrix\_4x4 **const** &m1,Matrix\_4x4 **const** &m2);
53. **friend** Matrix\_4x4 operator \* (Matrix\_4x4 **const** &m1,Matrix\_4x4 **const** &m2);
54. //重载 ^ 运算符为矩阵的i次幂（如果i为负数，如何处理？）
55. **friend** Matrix\_4x4 operator ^ (Matrix\_4x4 m1,**int** i);
56. //重载 [] 运算符，实现双下标方式访问矩阵元素
57. **const** **double** \* operator[] (**const** **int** i) **const** {**return** matrix[i];}
58. **double** \* operator[] (**const** **int** i){**return** matrix[i];}
59. // 重载输入<< 和输出 >>
60. **friend** ostream& operator <<(ostream& out\_put, **const** Matrix\_4x4 &m);
61. **friend** istream& operator >>(istream& in\_put, Matrix\_4x4 &m);
62. Matrix\_4x4 inverse();   //矩阵求逆，不改变当前矩阵值，返回逆矩阵
63. Matrix\_4x4 transpose(); //矩阵转置，不改变当前矩阵值，返回转置矩阵
64. **double** determinant();//矩阵求行列式，不改变当前矩阵值，返回行列式的值
65. };
67. Matrix\_4x4 **const** & Matrix\_4x4::operator = (Matrix\_4x4 **const** &m)
68. {
69. **for**(**int** i=0;i<4;i++)
70. {
71. **for**(**int** j=0;j<4;j++)
72. {
73. matrix[i][j]=m[i][j];
74. }
75. }
76. **return** \***this**;
77. }
79. Matrix\_4x4 **const** & Matrix\_4x4::operator = (**const** **double** m[4][4])
80. {
81. **for**(**int** i=0;i<4;i++)
82. {
83. **for**(**int** j=0;j<4;j++)
84. {
85. matrix[i][j]=m[i][j];
86. }
87. }
88. **return** \***this**;
89. }
91. Matrix\_4x4 operator + (**const** Matrix\_4x4 &m1,**const** Matrix\_4x4 &m2)
92. {
93. Matrix\_4x4 temp;
94. **for**(**int** i=0;i<4;i++)
95. {
96. **for**(**int** j=0;j<4;j++)
97. {
98. temp[i][j]=m1[i][j]+m2[i][j];
99. }
100. }
101. **return** temp;
102. }
104. Matrix\_4x4 operator - (**const** Matrix\_4x4 &m1,**const** Matrix\_4x4 &m2)
105. {
106. Matrix\_4x4 temp;
107. **for**(**int** i=0;i<4;i++)
108. {
109. **for**(**int** j=0;j<4;j++)
110. {
111. temp[i][j]=m1[i][j]-m2[i][j];
112. }
113. }
114. **return** temp;
115. }
117. Matrix\_4x4 operator \* (**const** Matrix\_4x4 &m1,**const** Matrix\_4x4 &m2)
118. {
119. Matrix\_4x4 temp;
120. **for**(**int** i=0;i<4;i++)
121. {
122. **for**(**int** j=0;j<4;j++)
123. {
124. temp[i][j]=0;
125. **for**(**int** n=0;n<4;n++)
126. {
127. temp[i][j]+=m1[i][n]\*m2[n][j];
128. }
129. }
130. }
131. **return** temp;
132. }
134. Matrix\_4x4 operator ^ (Matrix\_4x4 m,**int** n)
135. {
136. Matrix\_4x4 temp;
137. **for** (**int** i = 0; i < n; i++)
138. {
139. **for** (**int** j = 0; j < n; j++)
140. {
141. temp[i][j] = (i == j) ? 1 : 0;
142. }
143. }
144. **if**(n==0)
145. {
146. **return** temp;
147. }
148. **else** **if**(n>0)
149. {
150. **for**(**int** i=0;i<n;i++)
151. {
152. temp=temp\*m;
153. }
154. }
155. **else**
156. {
157. **for**(**int** i=0;i<-n;i++)
158. {
159. temp=temp\*m;
160. }
161. temp.inverse();
162. }
163. **return** temp;
164. }
166. ostream& operator <<(ostream& out\_put, **const** Matrix\_4x4 &m)
167. {
168. **for**(**int** i=0;i<4;i++)
169. {
170. **for**(**int** j=0;j<4;j++)
171. {
172. **if**(m[i][j]>1e-6)
173. {
174. out\_put<<m[i][j]<<"     ";
175. }
176. **else**
177. {
178. out\_put<<0<<"     ";
179. }
180. }
181. out\_put<<endl;
182. }
183. **return** out\_put;
184. }
186. istream& operator >>(istream& in\_put, Matrix\_4x4 &m)
187. {
188. **for**(**int** i=0;i<4;i++)
189. {
190. **for**(**int** j=0;j<4;j++)
191. {
192. in\_put>>m[i][j];
193. }
194. }
195. **return** in\_put;
196. }
198. Matrix\_4x4 Matrix\_4x4::inverse()
199. {
200. **int** i, j, k;
201. **double** max\_, temp;
202. Matrix\_4x4 t(matrix);//临时矩阵
203. Matrix\_4x4 B;//逆矩阵
204. //将A矩阵存放在临时矩阵t[n][n]中
205. //初始化B矩阵为单位阵
206. **for** (i = 0; i < 4; i++)
207. {
208. **for** (j = 0; j < 4; j++)
209. {
210. B[i][j] = (i == j) ? 1 : 0;
211. }
212. }
213. **for** (i = 0; i < 4; i++)
214. {
215. //寻找主元
216. max\_ = t[i][i];
217. k = i;
218. **for** (j = i+1; j < 4; j++)
219. {
220. **if** (fabs(t[j][i]) > fabs(max\_))
221. {
222. max\_ = t[j][i];
223. k = j;
224. }
225. }
226. //如果主元所在行不是第i行，进行行交换
227. **if** (k != i)
228. {
229. **for** (j = 0; j < 4; j++)
230. {
231. temp = t[i][j];
232. t[i][j] = t[k][j];
233. t[k][j] = temp;
234. //B伴随交换
235. temp = B[i][j];
236. B[i][j] = B[k][j];
237. B[k][j] = temp;
238. }
239. }
240. //判断主元是否为0, 若是, 则矩阵A不是满秩矩阵,不存在逆矩阵
241. **if** (t[i][i] == 0)
242. {
243. cout << "There is no inverse matrix!";
244. **return** B;
245. }
246. //消去A的第i列除去i行以外的各行元素
247. temp = t[i][i];
248. **for** (j = 0; j < 4; j++)
249. {
250. t[i][j] = t[i][j] / temp;        //主对角线上的元素变为1
251. B[i][j] = B[i][j] / temp;        //伴随计算
252. }
253. **for** (j = 0; j < 4; j++)        //第0行->第n行
254. {
255. **if** (j != i)                //不是第i行
256. {
257. temp = t[j][i];
258. **for** (k = 0; k < 4; k++)        //第j行元素 - i行元素\*j列i行元素
259. {
260. t[j][k] = t[j][k] - t[i][k]\*temp;
261. B[j][k] = B[j][k] - B[i][k]\*temp;
262. }
263. }
264. }
265. }
266. **return** B;
267. }
269. Matrix\_4x4 Matrix\_4x4::transpose()
270. {
271. **for** (**int** i = 0; i < 4; i++)
272. {
273. **for** (**int** j = i + 1; j < 4; j++)
274. {
275. swap(matrix[i][j], matrix[j][i]);
276. }
277. }
278. **return** matrix;
279. }
281. **void** swap(**double** &a, **double** &b)
282. {
283. **double** temp = a;
284. a = b;
285. b = temp;
286. **return**;
287. }
289. **double** Matrix\_4x4::determinant()
290. {
291. **int** num1,num2,index; // 初始化
292. **double** det=1;
293. **double** total=1;
294. **int** temp[5];
295. //循环遍历对角元
296. **for**(**int** i = 0; i < 4; i++)
297. {
298. index = i; // 初始化索引
299. //找寻非0值索引
300. **while**(matrix[index][i] == 0 && index < 4)
301. {
302. index++;
303. }
304. **if**(index == 4) // 如果这里没有非0元
305. {
306. // 行列式为0
307. **continue**;
308. }
309. **if**(index != i)
310. {
311. //交换行
312. **for**(**int** j = 0; j < 4; j++)
313. {
314. swap(matrix[index][j],matrix[i][j]);
315. }
316. //行列式的正负会变如果交换两行
317. det = det\*pow(-1,index-i);
318. }
319. **for**(**int** j = 0; j < 4; j++)
320. {
321. temp[j] = matrix[i][j];
322. }
323. //遍历每一行
324. **for**(**int** j = i+1; j < 4; j++)
325. {
326. num1 = temp[i]; //对角元的值
327. num2 = matrix[j][i]; //下一行的值
328. **for**(**int** k = 0; k < 4; k++)
329. {
330. matrix[j][k] = (num1 \* matrix[j][k]) - (num2 \* temp[k]);
331. }
332. total = total \* num1; // Det(kA)=kDet(A);
333. }
334. }
335. //乘起来对角元
336. **for**(**int** i = 0; i < 4; i++)
337. {
338. det = det \* matrix[i][i];
339. }
340. **return** (det/total); //Det(kA)/k=Det(A);
341. }
343. **int** main()
344. {
345. **double** m[4][4]={{1,2,3,4},{8,6,7,9},{4,10,-4,12},{-13,14,45,28}};
346. Matrix\_4x4 a;
347. Matrix\_4x4 b(a);
348. Matrix\_4x4 c(m);
349. Matrix\_4x4 d;
350. Matrix\_4x4 e;
351. **double** temp;
353. **for**(**int** i=0;i<4;i++)
354. {
355. **for**(**int** j=0;j<4;j++)
356. {
357. d[i][j]=m[i][j];
358. }
359. }
361. cout<<"a:"<<endl<<a<<endl;
362. cout<<"b(a)"<<endl<<b<<endl;
363. cout<<"c(m)"<<endl<<c<<endl;
364. cout<<"d"<<endl<<d<<endl;
366. d=a+c;
367. cout<<"d=a+c:"<<endl<<d<<endl;
369. d=d-a;
370. cout<<"d=d-a"<<endl<<d<<endl;
372. e=d.inverse();
373. cout<<"e:"<<endl<<e<<endl;
375. d=d\*e;
376. cout<<"d:"<<endl<<d<<endl;
378. d=a^3;
379. cout<<"d=a^3"<<endl<<a<<endl;
381. d=c.transpose();
382. cout<<"d=c.transpose()"<<endl<<d<<endl;
384. temp=c.determinant();
385. cout<<"temp=c.determinant"<<endl<<temp<<endl;
387. **return** 0;
388. }







3、矩阵的幂预算中，负次幂如何处理，请说明。

对于-i次幂，先算矩阵的i次幂，再求逆。

4、选做题：解释为什么当前代码可以实现对象名双下标访问对象内数组元素，以及有无const的区别

const double \* operator[] (const int i) const {return matrix[i];}

double \* operator[] (const int i){return matrix[i];}

第一个[]返回的是一个行指针，再次[]索引即把每一行当成一个新的数组，再次访问得到的是一个具体的元素，即实现了双下标访问。

**三 实验总结与建议**

（总结实验实施过程，说明实验过程中遇到的问题与解决方案；提出实验环节的建议）

输出矩阵的时候会有很小的例如10的-12次方的元素，所以在输出的时候加入了一个对元素大小的判断，小于10的-6次方的一律按0处理。