《数据结构与算法》实验报告

题目：一元多项式的基本运算

分工：实验2

姓名：张富利

学号：1507020329

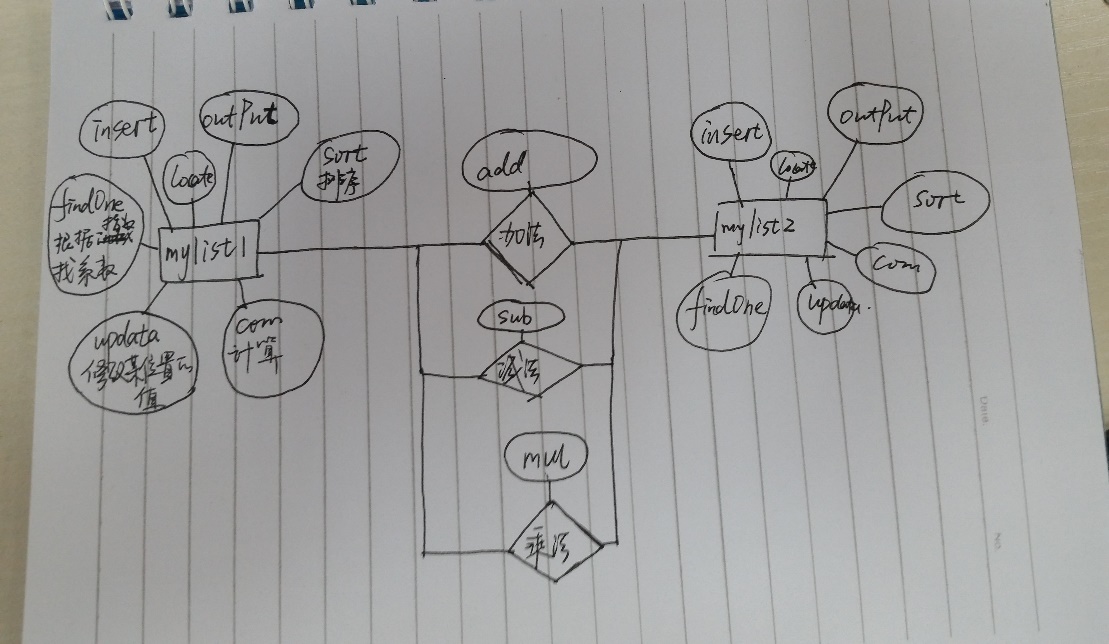
团队成员： 张富利、杨强、陈子枫、王浩竣

2017年3月

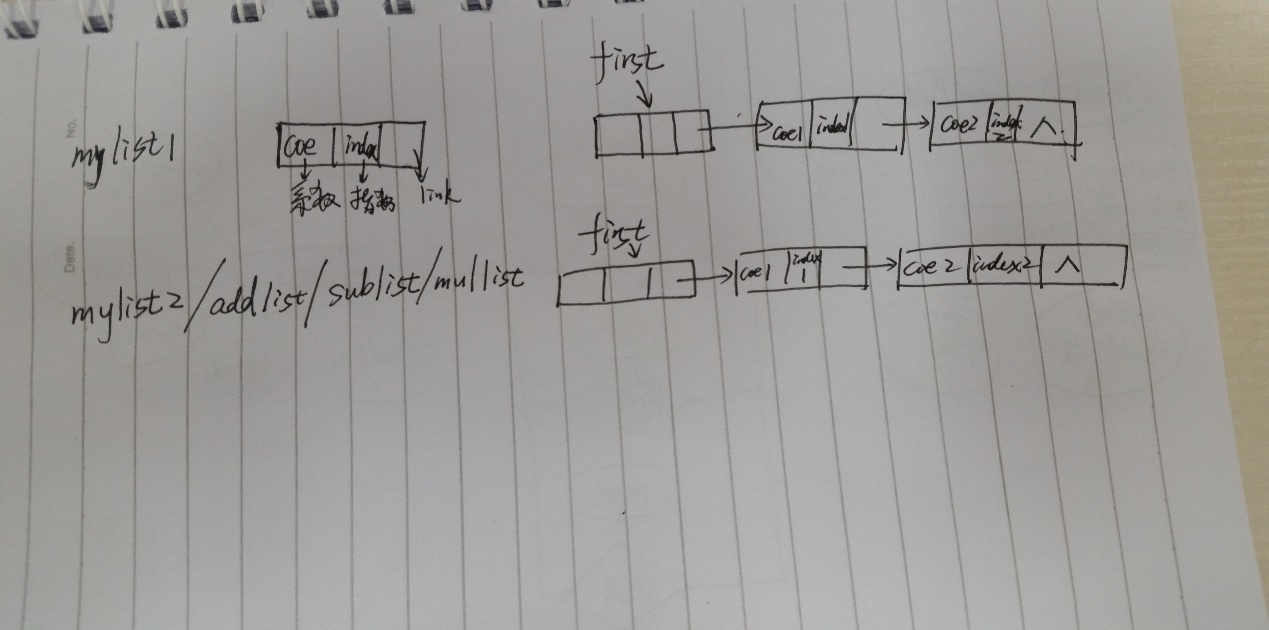
1. **分工任务描述**

整个实验分为定义链表、设计输出、加法、减法、乘法、求导、求值等7个模块，其中，我具体负责全部模块。主要实现此次试验的全部要求

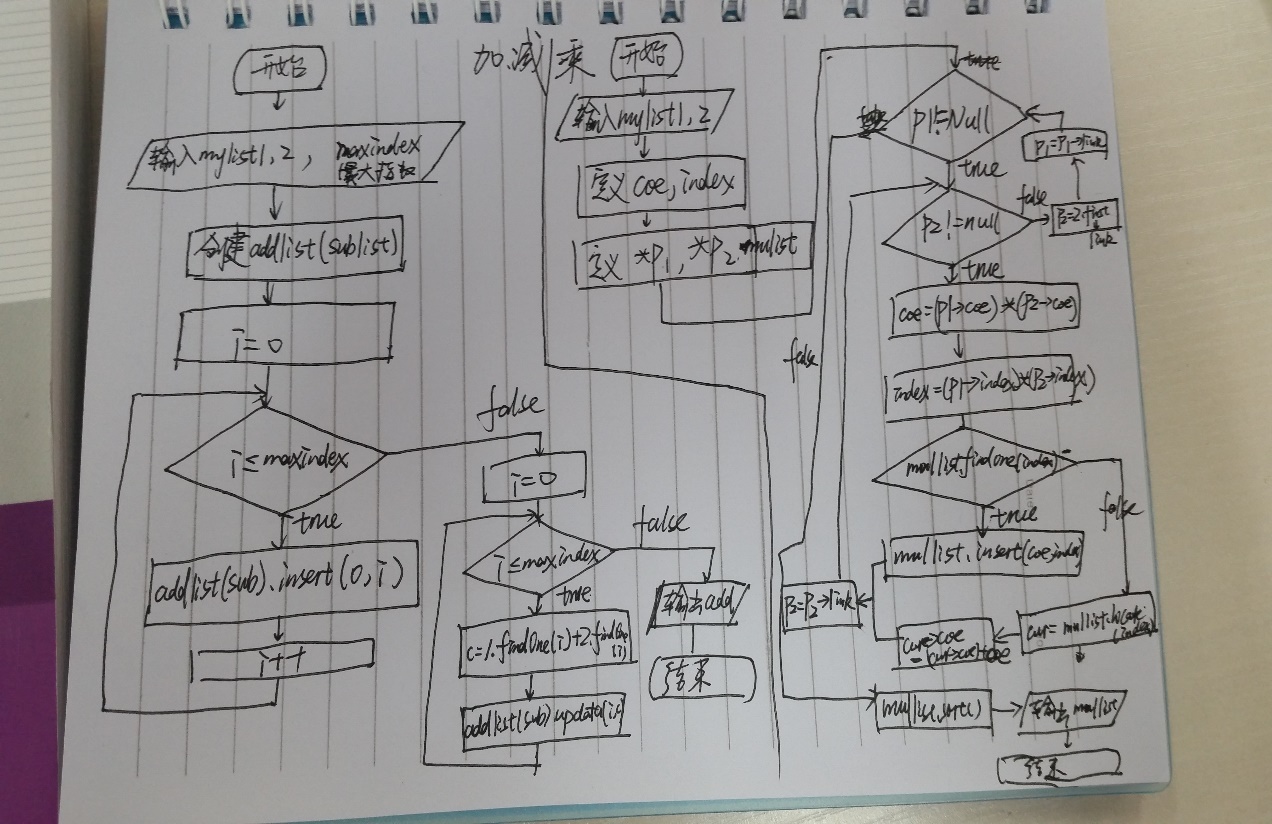
1. **逻辑结构设计**



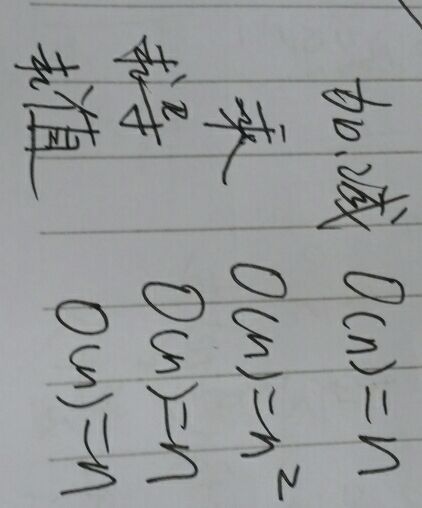
1. **物理结构设计**



1. **程序流程图**

（画出主要函数的程序流程图。）****

1. **算法复杂度分析**

****

1. **心得体会**

对于数据结构这门课程我觉得现在谈认识还有点早，但是我觉得对于指、链表的掌握是非常有必要的。对于我而言，指针其实学的并不是很好，但是通过画图我可以更好的掌握链表。

通过独立完成这次实验，我觉得对链表有了更深层的认识，链表的变形十分丰富，对于具体的项目要选择不同的链表。

1. **附录**
2. #include <iostream>
3. #include "stdlib.h"
4. #include <iomanip>
5. **using** **namespace** std;
6. **class** List;
7. **class** LinkNode{
8. friend **class** List;
9. **public**:
10. **int** coe;
11. **int** index;
12. LinkNode \*link;
13. LinkNode(LinkNode \*ptr = NULL){link = ptr;}
14. LinkNode(**int** c,**int** i,LinkNode \*ptr = NULL){
15. coe = c;
16. index = i;
17. link = ptr;
18. }
19. ~LinkNode();
20. };
21. **class** List{
22. **public**:
23. LinkNode \*first;
24. **int** length;
25. List(){
26. first = **new** LinkNode();
27. length = 0;
28. }
29. **void** inSert(**int** coe,**int** index);  //后插法插入
30. **void** outPut();
31. **void** outPut(**int** x);
32. **void** Der();
33. **int** findOne(**int** x);  //找到index=x，返回其coe,没有返回0
34. LinkNode\* locate(**int** x);  //找到index=x的位置,返回其地址,没有返回NULL
35. **void** updata (**int** i,**int** c); //修改某位置的值
36. **void** com(**float** x); //计算结果
37. **void** sort();
38. };
39. **void** List::inSert(**int** coe,**int** index){
40. LinkNode \*newNode = **new** LinkNode(coe,index);
41. LinkNode \*p = first;
42. **for**(**int** i=0;i<=length;i++){
43. **if**(p->link ==NULL){
44. p->link = newNode;
45. p = newNode;
46. }
47. p = p ->link;
48. }
49. length++;
50. }
51. **void** List::outPut(){
52. LinkNode \*p = first->link;
53. **int** log = 0; //记录第一次
54. **while**(p!=NULL){
55. **if**(p->coe!=0){
56. **if**(log!=0){
57. **if**(p->index == 1){
58. **if**(p->coe==1){
59. cout<<"+x";
60. }**else** **if**((p->coe)<0){
61. cout<<p->coe<<"x";
62. }**else**{
63. cout<<"+"<<p->coe<<"x";
64. }
65. }**else** **if**(p->index==0){
66. **if**(p->coe==1){
67. cout<<"+"<<p->coe;
68. }**else** **if**((p->coe)<0){
69. cout<<p->coe;
70. }**else**{
71. cout<<"+"<<p->coe;
72. }
73. }**else**{
74. **if**(p->coe==1){
75. cout<<"+x^"<<p->index;
76. }**else** **if**((p->coe)<0){
77. cout<<p->coe<<"x^"<<p->index;
78. }**else**{
79. cout<<"+"<<p->coe<<"x^"<<p->index;
80. }
81. }
82. }**else**{
83. **if**(p->index == 1){
84. **if**(p->coe==1){
85. cout<<"x";
86. }**else**{
87. cout<<p->coe<<"x";
88. }
89. }**else** **if**(p->index==0){
90. cout<<p->coe;
91. }**else**{
92. **if**(p->coe==1){
93. cout<<"x^"<<p->index;
94. }**else**{
95. cout<<p->coe<<"x^"<<p->index;
96. }
97. }
98. }
99. log++;
100. }
101. p= p->link;
102. }
103. }
104. **void** List::sort(){
105. **int** indexItem,coeItem;
106. **for**(**int** i=0;i<length;i++){
107. LinkNode \*p = first->link;
108. **while**(p->link!=NULL){
109. **if**( (p->index) > (p->link->index)){
110. indexItem = p->index;
111. p->index = p->link->index;
112. p->link->index = indexItem;
114. coeItem = p->coe;
115. p->coe = p->link->coe;
116. p->link->coe = coeItem;
117. }
118. p = p->link;
119. }
120. }
121. }
122. **void** List::Der(){
123. LinkNode \*p = first->link;
124. **int** log = 0;
125. **while**(p!=NULL){
126. **if**(p->coe!=0){
127. **if**(log!=0){
128. **if**(p->index==1){
129. **if**(p->coe==1){
130. cout<<"+"<<p->coe\*p->index;
131. }**else** **if**(p->coe<0){
132. cout<<p->coe\*p->index;
133. }**else**{
134. cout<<"+"<<p->coe\*p->index;
135. }
136. }**else** **if**(p->index == 2){
137. **if**(p->coe==1){
138. cout<<"+"<<p->coe\*p->index<<"x";
139. }**else** **if**(p->coe<0){
140. cout<<p->coe\*p->index<<"x";
141. }**else**{
142. cout<<"+"<<p->coe\*p->index<<"x";
143. }
144. }**else** **if**(p->index==0){
145. }**else**{
146. **if**(p->coe==1){
147. cout<<"+"<<p->coe\*p->index<<"x^"<<p->index-1;
148. }**else** **if**(p->coe<0){
149. cout<<p->coe\*p->index<<"x^"<<p->index-1;
150. }**else**{
151. cout<<"+"<<p->coe\*p->index<<"x^"<<p->index-1;
152. }
153. }
154. }**else**{
155. **if**(p->index == 1){
156. **if**(p->coe==1){
157. cout<<p->coe\*p->index;
158. }**else**{
159. cout<<p->coe\*p->index;
160. }
161. }**else** **if**(p->index == 2){
162. **if**(p->coe==1){
163. cout<<p->coe\*p->index<<"x";
164. }**else**{
165. cout<<p->coe\*p->index<<"x";
166. }
167. }**else** **if**(p->index==0){
168. log--;
169. }**else**{
170. **if**(p->coe==1){
171. cout<<p->coe\*p->index<<"x^"<<p->index-1;
172. }**else**{
173. cout<<p->coe\*p->index<<"x^"<<p->index-1;
174. }
175. }
177. }
178. log++;
179. }
180. p= p->link;
181. }
182. }
183. **int** List::findOne(**int** x){
184. LinkNode \*p = first->link;
185. **while**(p!=NULL){
186. **if**(p->index == x){
187. **return** p->coe;
188. }
189. p = p->link;
190. }
191. **return** 0;
192. }
193. LinkNode\* List::locate(**int** x){
194. LinkNode \*p = first->link;
195. **while**(p!=NULL){
196. **if**(p->index == x){
197. **return** p;
198. }
199. p = p->link;
200. }
201. **return** NULL;
202. }
203. **void** List::updata (**int** i,**int** c){
204. LinkNode \*p = locate(i);
205. **if**(p!=NULL){
206. p->coe = c;
207. }
208. }
209. **void** List::com(**float** x){
210. **float** val = 1.0,sum=0.0;
211. LinkNode \*p = first->link;
212. **while**(p!=NULL){
213. val = 1.0;
214. val = val\*(p->coe);
215. **for**(**int** i=0;i<p->index;i++){
216. val = val\*x;
217. }
218. sum=sum+val;
219. p=p->link;
220. }
221. cout<<setiosflags(ios::**fixed**)<<setprecision(2)<<sum<<endl;
222. }
223. **void** add(List list1,List list2,**int** maxindex){
224. List addlist;
225. **int** i;
226. **for**(i=0;i<=maxindex ;i++){
227. addlist.inSert(0,i);
228. }
229. **for**(i=0;i<=maxindex;i++){
230. addlist.updata(i,list1.findOne(i)+list2.findOne(i));
231. }
232. addlist.outPut();
233. }
234. **void** sub(List list1,List list2,**int** maxindex){
235. List sublist;
236. **int** i;
237. **for**(i=0;i<=maxindex ;i++){
238. sublist.inSert(0,i);
239. }
240. **for**(i=0;i<=maxindex;i++){
241. sublist.updata(i,list1.findOne(i)-list2.findOne(i));
242. }
243. sublist.outPut();
244. }
245. **void** mul(List list1,List list2){
246. List mullist;
247. **int** coe,index;
248. LinkNode \*p1 = (list1.first)->link;
249. LinkNode \*p2 = (list2.first)->link;
250. LinkNode \*cur;
251. **while**(p1!=NULL){
252. **while**(p2!=NULL){
253. coe = (p1->coe)\*(p2->coe);
254. index = (p1->index)+(p2->index);
255. **if**(mullist.findOne(index)==0){
256. mullist.inSert(coe,index);
257. }**else**{
258. cur = mullist.locate(index);
259. cur->coe = (cur->coe)+coe;
260. }
261. p2 = p2->link;
262. }
263. p2 = (list2.first)->link;
264. p1 = p1->link;
265. }
266. //排序
267. mullist.sort();
268. mullist.outPut();
269. }
271. **int** main(){
272. //C,P,S,M,D,V,C
273. List mylist1 ,mylist2;
274. **int** maxindex = 0;
275. **int** coe,index;
276. **float** data;
277. **char** commend;
278. **while**(1){
279. cin>>commend;
280. **if**(commend == 'C'){
281. **while**(**true**){
282. cin>>coe;
283. **if**(coe == 0){
284. **break**;
285. }**else**{
286. cin>>index;
287. **if**(index>maxindex){maxindex = index;}
288. mylist1.inSert(coe,index);
289. }
290. }
291. **while**(**true**){
292. cin>>coe;
293. **if**(coe == 0){
294. **break**;
295. }**else**{
296. cin>>index;
297. **if**(index>maxindex){maxindex = index;}
298. mylist2.inSert(coe,index);
299. }
300. }
301. }**else** **if**(commend == 'P'){
302. cout<<"C(x)=";
303. add(mylist1,mylist2,maxindex);
304. cout<<endl;
305. }**else** **if**(commend == 'S'){
306. cout<<"C(x)=";
307. sub(mylist1,mylist2,maxindex);
308. cout<<endl;
309. }**else** **if**(commend == 'M'){
310. cout<<"C(x)=";
311. mul(mylist1,mylist2);
312. cout<<endl;
313. }**else** **if**(commend == 'D'){
314. cout<<"C(x)=";
315. mylist1.Der();
316. cout<<endl;
317. }**else** **if**(commend == 'V'){
318. cin>>data;
319. mylist1.com(data);
320. }**else**{
321. **break**;
322. }
323. }
324. **return** 0;
325. }