

《数据结构》上机实验报告

第 4 次上机

学号：

姓名：

学院： 信息科学学院

专业： 计算机科学

教师：

日期： 2018.9.21

1. 实验要求

1.上机之前应做好充分准备，认真思考所需的上机题目，提高上机效率。

2.独立上机输入和调试自己所编的程序，切忌抄袭、拷贝他人程序。

3.上机结束后，整理出实验报告。书写报告时，重点放在实验的方法、思路以及总结反思上，以达到巩固课堂学习、提高动手能力的目的。

1. 实验内容

合理运用栈，按照教材中的运算优先级，编程实现任意中缀算术表达式（可以只包含’+’、’-‘、’\*’、’/’等双目运算符、小括号和结束符）的求值运算。

1. 实验步骤（写出问题分析或者算法思路）

思路：按照ppt上的开双栈，使用一个存元素（OPND），一个存操作符(OPTR)实现中缀算法了。 使用带头节点的链表实现栈是很方便的，所以参考一些了ppt上的用链表表示栈的方法。对于操作数和操作符组成的算式，采用的方法是把算式先看成是字符串，然后参考了c++中对字符串操作的一些知识，把它们转化成了需要栈元素。然后就可以实现ppt和老师讲的中缀式子算法了。

课上和ppt里已经有关于栈的初始化、进栈、出栈、取顶、查空的思想方法和具体实现，这次也借鉴了很多。分别见initStack, Push,Pop, getTop和stackEmpty函数。注意到两个栈(OPND和OPTR的元素都是int类型的数，也就是说，两个栈共用了一个公共定义的Node，这主要是为了简化栈函数的写法——只需要一个统一的进栈、出栈等操作的函数而不必每个操作函数都两个栈各写一次，至于怎么用整数表示操作符，参见总结1)

栈的输出是用递归实现的，参见Print函数。

对字符串的操作式这样的：先getline读取整个算式，再一个字符一个字符地读取。遇到数字，则先看它有几位数（遍历），再把它转化成数字（参见trans函数）；遇到操作符，直接操作即可。只考虑了比较标准的正确输入的算式（算式只有小括号、括号是匹配的、不省略0、没有负数、没有空格、不省略乘号、模数不为0等）。

中缀表达式与后缀、前缀表达式最大的不同在于括号和操作符的优先级了。由于括号的加入，所以要先算在括号里的值。由于有操作符的优先级，所以要考虑运算的顺序。老师上课讲了计算中缀表达式，我觉得算法的核心在于这个表。



这个表非常有用。第一，它把中缀表达式中的左右括号看成和一般操作符（+-\*/等）同级的，从而简化了算法的复杂性。试想如果单独把括号先读出来，再由里到外地计算括号里的值，也可以实现算法，但是要多写一个对括号的处理。而通过使用这个表，把括号当成一般符号处理，简化了算法的复杂性。第二，它把中缀式中的操作符不同的运算优先级体现了出来，从而解决了中缀表达式中不同的运算顺序问题。这也解释了为什么括号也能出现在这个表中：括号也具有运算优先级，与中缀表达式中一般操作符非常类似。总的来说，通过这个表，中缀表达式能找到运算的正确顺序，从而和前缀、后缀表达式一样，能够利用栈解决运算的问题。

对操作符的优先级是用switch语句来实现的，参见priority函数。

具体的算法和ppt、老师讲的一样。

一共实现了+-\*/%^运算。除法是向下取整，幂运算用到了快速幂来加快速度，对0取模、除以0会报错。

1. 程序清单（源程序代码等）

main.h:

1. #include<iostream>
2. #include <stdio.h>
3. #include <malloc.h>
4. #include <fstream>
5. #include <string>
6. **using** **namespace** std;
7. **int** priority(**int** type,**const** **int** operate) //type=1 isp    type=2 icp
8. {
9. **if**(type==1)
10. {
11. **switch**(operate)
12. {
13. **case** '#':**return** 0;
14. **case** '(':**return** 1;
15. **case** '^':**return** 7;
16. **case** '\*':**return** 5;
17. **case** '/':**return** 5;
18. **case** '%':**return** 5;
19. **case** '+':**return** 3;
20. **case** '-':**return** 3;
21. **case** ')':**return** 8;
22. }
23. }
24. **else**
25. {
26. **switch**(operate)
27. {
28. **case** '#':**return** 0;
29. **case** '(':**return** 8;
30. **case** '^':**return** 6;
31. **case** '\*':**return** 4;
32. **case** '/':**return** 4;
33. **case** '%':**return** 4;
34. **case** '+':**return** 2;
35. **case** '-':**return** 2;
36. **case** ')':**return** 1;
37. }
38. }
39. }
40. **int** trans(string a,**int** len) //字符串转数字
41. {
42. **int** ans=0;
43. **for**(**int** i=0;i<len;i++)
44. {
45. ans\*=10;
46. ans+=a[i]-'0';
47. }
48. **return** ans;
49. }
50. **typedef** **struct** node {            //栈定义
51. **int** Data;
52. **struct** node \*link;            //结点链接指针
53. } LinkNode, \*LinkStack;      //链式栈
54. **bool** Push ( LinkStack& S,**int** tmpD);
55. **void** initStack ( LinkStack& S )
56. {   //栈初始化
57. S = NULL;          //置空栈
58. Push(S,'#');
59. }
61. **bool** stackEmpty ( LinkStack& S )
62. {    //判栈空否
63. **return** S == NULL;
64. }
65. **int** power(**int** a,**int** b)
66. {
67. **int** ans=1;
68. **while**(b)
69. {
70. **if**(b&1)
71. {
72. ans\*=a;
73. }
74. b/=2;
75. a=a\*a;
76. }
77. **return** ans;
78. }
79. **bool** Push ( LinkStack& S,**int** tmpD)
80. {  //进栈
81. LinkNode \*p = (LinkNode \*) malloc ( **sizeof**(LinkNode ));                   //创建新结点
82. **if** ( p == NULL )
83. { printf ("结点创建失败!\n");  exit (1); }
84. p->Data = tmpD;                     //结点赋值
85. p->link = S;
86. S = p;
87. //printf("In : %d\n",tmpD);
88. **return** **true**;     //链入栈顶
89. }
91. **bool** Pop ( LinkStack& S, **int** & x ) {    //退栈
92. **if** ( stackEmpty(S) ) **return** **false**;
93. LinkNode \* p = S;
94. x = p->Data;
95. S = p->link;        //摘下原栈顶
96. free (p);
98. //printf("Out : %d\n",x);
99. **return** **true**;       //释放原栈顶结点
100. }
102. **bool** getTop ( LinkStack& S, **int**& x ) {
103. //看栈顶元素
104. **if** ( stackEmpty (S) ) **return** **false**;
105. x = S->Data;  **return** **true**;
106. }
108. **void** Print(LinkStack &S,**int** type)
109. {
110. **if**(type==1)
111. {
112. **if**(S==NULL)
113. **return**;
114. Print(S->link,type);
115. **if**(S->Data!=35)
116. printf("\_%d",S->Data);
117. }
118. **else**
119. {
120. **if**(S==NULL)
121. **return**;
122. Print(S->link,type);
123. **if**(S->Data!='#')
124. printf("\_%c",S->Data);
125. **else**
126. printf("%c",S->Data);
127. }
128. }
130. **int** main ()
131. {
132. **int** ans;
133. ifstream OpenFile("test.txt");    //两种文件有两种不同的定义
134. string str;
136. **while**(getline(OpenFile,str))
137. {
138. LinkStack OPTR,OPND;
139. initStack(OPTR);
140. initStack(OPND);
141. cout<<"算式是"<<str<<endl;
142. //        str[str.length()+1]='\0';
143. //        str[str.length()]='#';
144. **int** i=0;
145. **while**(1)
146. {
147. **if**(str[i]=='\0')
148. str[i]='#';
149. //printf("i=%d\n",i);
150. **if**(str[i]>='0' && str[i] <='9')
151. {
152. **int** j;
153. **for**(j=0;str[i+j];j++)
154. {
155. **if**(str[i+j]>='0' && str[i+j]<='9')
156. **continue**;
157. **break**;
158. }
159. string tmp=str.substr(i,j);
160. //cout<<tmp<<endl;
161. **int** num=trans(tmp,j);
162. Push(OPND,num);
163. printf("Case:遇到数字\t读取数值为 [%d]\t直接进OPND栈\n",num);
164. printf("OPND:#");
165. Print(OPND,1);
166. printf("\n");
167. //cout<<num<<endl;

170. i+=j;
171. }
172. **else**
173. {
175. **int** top;
176. Pop(OPTR,top);
177. printf("Case:遇到操作符\t读取操作符为 [%c]\t与OPTR的top元素[%c]比较\t结果是",str[i],top);
178. //printf("%c %d vs %c %d\n",str[i],str[i],top,top);
179. **if**(priority(1,top) < priority(2,str[i]))
180. {
181. printf("直接进OPTR栈\n");
182. Push(OPTR,top);
183. Push(OPTR,str[i]);
184. i++;
185. }
186. **else** **if**(priority(1,top) > priority(2,str[i]))
187. {
188. **int** num1,num2;
189. Pop(OPND,num2);
190. Pop(OPND,num1);
191. **switch**(top)
192. {
193. **case** '+':num1=num1+num2;**break**;
194. **case** '^':num1=power(num1,num2);**break**;
195. **case** '\*':num1=num1\*num2;**break**;
196. **case** '/':num1=num1/num2;**break**;
197. **case** '%':num1=num1%num2;**break**;
198. **case** '-':num1=num1-num2;**break**;
199. }
200. Push(OPND,num1);
201. printf("取两个数进行%c运算 后进入OPND栈\n",top);
202. printf("OPND:#");
203. Print(OPND,1);
204. printf("\n");
205. }
206. **else**
207. {
208. **if**(str[i]==')')
209. {
210. printf("左右括号相互抵消后 继续\n");
211. i++;
212. **continue**;
213. }
214. **else**
215. {
216. printf("都是#号相互抵消后 停止\n");
217. **break**;
218. }
219. }
220. printf("OPTR:");
221. Print(OPTR,2);
222. printf("\n");
223. }
224. }
225. Pop(OPND,ans);
226. printf("程序答案是:");
227. cout<<str;
228. printf("=%d\n--------\n",ans);
229. }
231. printf("给出\"test.txt\"里算式的正确答案:\n");
232. printf("正确答案是:2^(3+7)+10000/20=%d\n",power(2,(3+7))+10000/20);
233. printf("正确答案是:((11-22)\*44)\*(55-66)=%d\n",((11-22)\*44)\*(55-66));
234. printf("正确答案是:(3+5)%3\*4=%d\n",(3+5)%3\*4);
235. **return** 0;
236. }

test.txt:

2^(3+7)+10000/20

((11-22)\*44)\*(55-66)

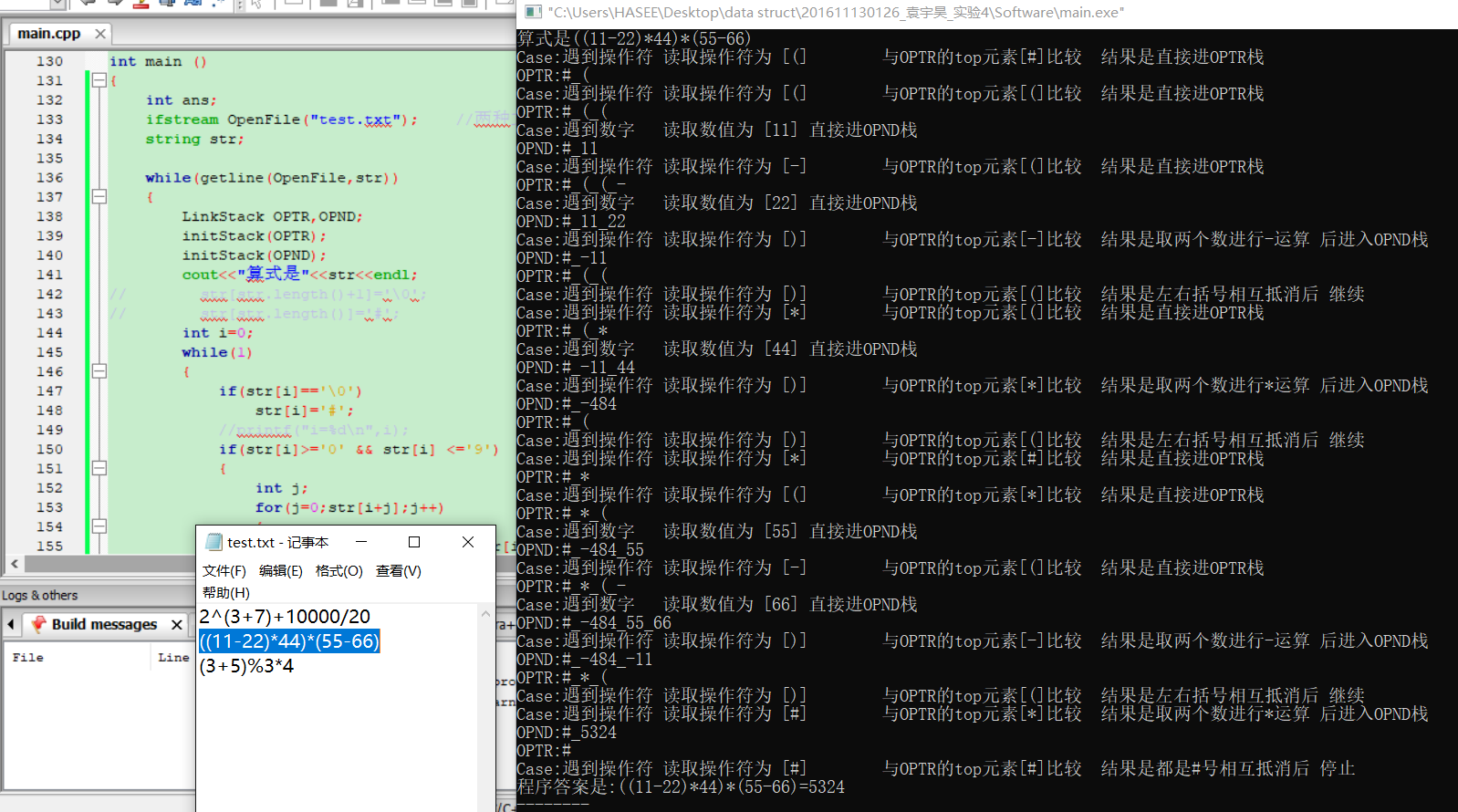
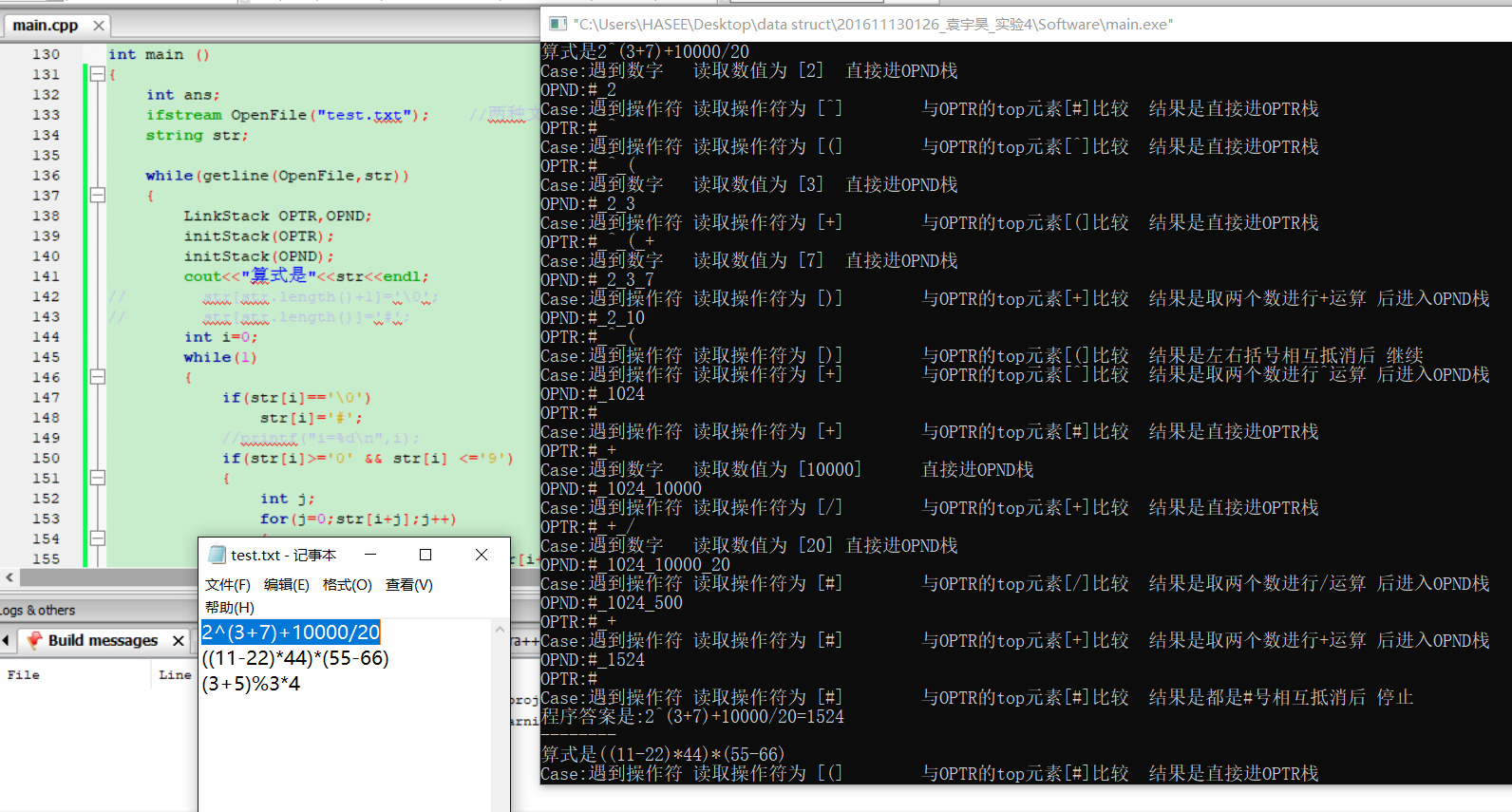
(1+(2-(3+(4\*5))))

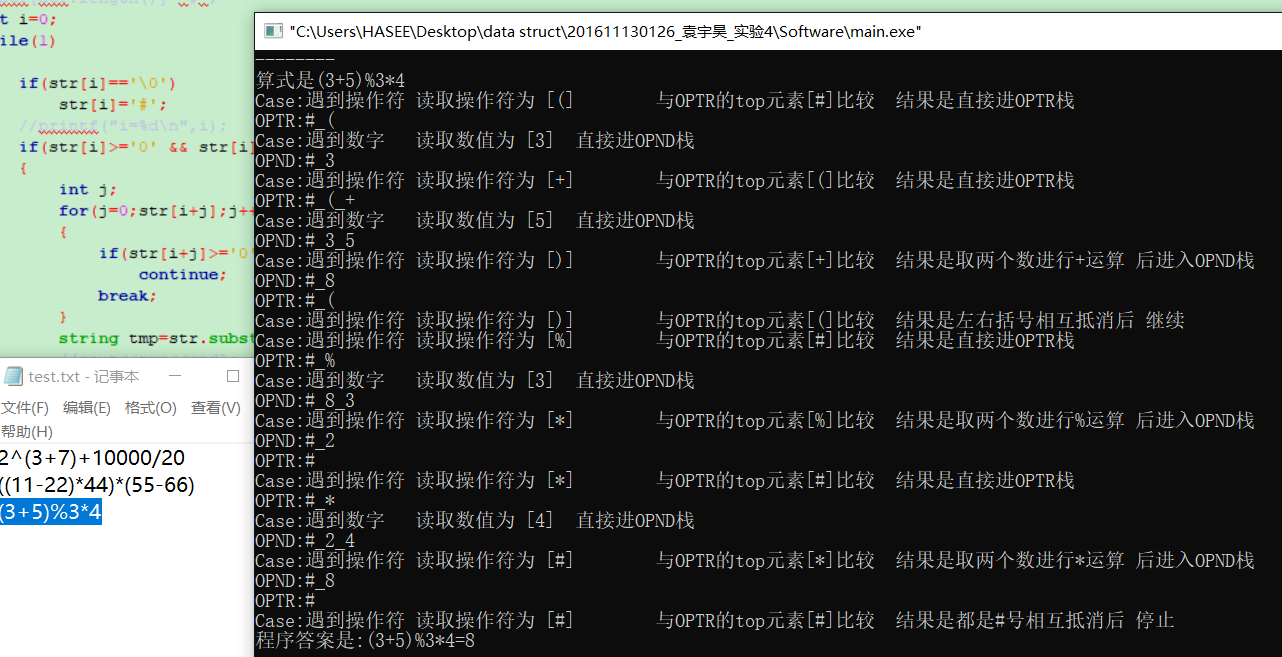
1. 运行结果（程序运行时的结果说明或运行截图等）

使用了ifstream库来读入测试数据“tset.txt”。

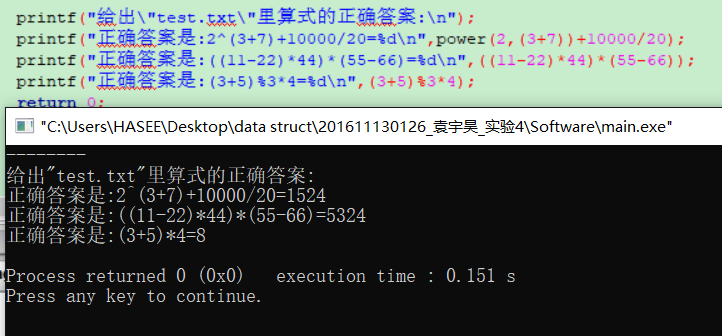
注意第1组样例用到了幂运算（^）。

第3组样例用到了取模运算（%）





给出正确答案



1. 总结（实验中遇到的问题、取得的经验、感想等）

1 关于操作符和操作数共用一个Node定义，是因为操作符占一个char类型，操作数占一个int类型，而char类型和int类型之间的互换是非常友好的，所以统一用了一个int Data来定义它们，这在程序实现完成后看来没什么问题。

2 由于使用的是链表建栈，所以基本上不会溢出。

3 输出栈时使用了递归函数，输出的顺序与递归函数里的递归调用和printf函数顺序一致。

4 每次读新算式都要初始化数据。

5 太大的数可能会超过int的范围，可以改int为long long。