**数据结构实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号-姓名 | 桑龙龙-20030540015 | 实验时间 | 2020 年 10 月31 日 |
| 诚信声明 | 本实验及实验报告所写内容为本人所作，没有抄袭。 | | |
| 实验题目 | 题目一、字符串基本运算  题目二、文学研究助手 | | |
| 实验过程中遇到的主要问题 | 在将字符串基本操作函数封装到类中时，在排序操作后，发现会出现访问越界的问题，最后经过检查发现C++自带的sort函数会有一个拷贝及销毁过程，在String的析构函数中释放了已经分配内存，所以在之后再访问时就会出现越界。另一个问题就是类中的const修饰的函数，经过学习才明白const修饰的函数只能访问const修饰的函数及变量。 | | |
| 实验小结 | 本次试验进行了字符串函数相关的学习，自己对字符串及C++的一些语法有了更深刻的理解与认识 | | |
| 数据结构  （自定义数据类型） | 字符串的基本操作封装到String类中 | | |
| 主要算法  （或算法说明） | 1. **class** String{ 2. **private**: 3. **int** length=0;   //字符串实际大小 4. **int** size=0; //字符串拥有空间大小 5. **char**\* arr=NULL;//字符串数据存放 6. **int** add = 20;//字符串额外扩容空间 7. **public**: 9. //#######构造函数区######## 10. //构造函数1： 11. //不含任何内容的空字符串 12. String(){ 14. } 15. //构造函数2： 16. //将字符串大小初始化为\_size 17. String(**int** \_size){ 19. length = \_size; 20. size = \_size+add; 21. arr = (**char**\*)malloc(**sizeof**(**char**) \* size); 22. arr[length] = 0; 23. } 24. //构造函数3： 25. //字符串大小初始化为\_size 26. //内容都为val 27. String(**int** \_size, **char** val){ 29. length = \_size; 30. size = \_size + add; 31. arr = (**char**\*)malloc(**sizeof**(**char**) \* size); 32. **for**(**int** i=0;i<length;i++) arr[i]=val; 33. arr[length] = 0; 34. } 35. //构造函数4： 36. //将字符串初初始化为const char\*:b 37. String(**const** **char**\* b){ 38. length = strlen(b); 39. size = length + add; 40. arr = (**char**\*)malloc(**sizeof**(**char**) \* size); 41. **for**(**int** i = 0; i < length; i++) arr[i] = b[i]; 42. arr[length] = 0; 43. } 44. //构造函数5： 45. //将字符串初初始化为char\*:b 46. String(**char**\* b){ 47. length = strlen(b); 48. size = length+add; 49. arr = (**char**\*)malloc(**sizeof**(**char**) \* size); 50. **for**(**int** i = 0; i < length; i++) arr[i] = b[i]; 51. arr[length] = 0; 52. } 53. //拷贝构造函数: 54. //将字符串初初始化为String:b 56. String(**const** String &b){ 57. length=b.length; 58. size=b.size; 59. arr=b.arr; 60. add=b.add; 61. } 63. ~String(){ 64. //free(arr); 65. } 67. //######辅助函数区########## 69. //字符串长度函数 70. **int** strlen(**const** **char**\* a){ 71. **int** cnt=0; 72. **while**(\*a) a++,cnt++; 73. **return** cnt; 74. } 76. //动态扩容函数： 77. //如果当前字符串的大小 78. //大于已分配的实际空间，进行动态扩容 79. **void** expand(**int** new\_size){ 80. **if**(new\_size <= size) **return**; 81. size=add+new\_size; 82. arr = (**char**\*)realloc(arr, **sizeof**(**char**) \* size); 83. } 84. //重设大小函数： 85. //改变字符串的大小（非实际占有空间） 86. **void** resize(**int** \_length){ 87. **if**(\_length <= length){ 88. length = \_length; 89. arr[length] = 0; 90. }**else**{ 91. expand(\_length); 92. length = \_length; 93. } 94. } 95. //尾插函数： 96. //向字符串最后插入一个字符 97. **void** push\_back(**char** val){ 98. expand(length + 2); 99. arr[length++] = val; 100. arr[length] = 0; 101. } 102. //尾删函数： 103. //删除字符串最后一个字符 104. **void** pop\_back(){ 105. length -= length == 0 ? 0 : 1; 106. arr[length] = 0; 107. } 108. //返回字符串的指针，用于输出 109. **char**\* c\_str(){ 110. **return** arr; 111. } 112. //返回字符串的指针，用于输出 113. **char**\* data()**const**{ 114. **return** arr; 115. } 116. //重载[]，能够通过下标访问字符 117. **char**& operator[](**int** index)**const**{ 118. **return** arr[index]; 119. } 121. **bool** operator ==(**const** String& b)**const**{ 122. **return** StrCmp(b)==0; 123. } 124. **bool** operator <(**const** String& b)**const**{ 125. **return** StrCmp(b)==-1; 126. } 127. **bool** operator <=(**const** String& b)**const**{ 128. **return** StrCmp(b)<=0; 129. } 130. **bool** operator >(**const** String& b)**const**{ 131. **return** StrCmp(b)==1; 132. } 133. **bool** operator >=(**const** String& b)**const**{ 134. **return** StrCmp(b)>=0; 135. }   139. //#######功能区######### 140. //1、长度函数: 141. **int** StrLength()**const**{//1 142. **return** length; 143. } 145. //2、StrAssign赋值函数 146. //可以接受的字符串类型： 147. //String，char\*两种 148. **void** StrAssign(**char**\* b, **int** len){ 149. expand(len); 150. **for**(**int** i = 0; i < len; i++) arr[i] = b[i]; 151. length = len; 152. arr[length] = 0; 153. } 154. **void** StrAssign(**char**\* b){ 155. StrAssign(b, strlen(b)); 156. } 157. **void** StrAssign(**const** String & b){//2 158. StrAssign(b.data(), b.StrLength()); 159. } 160. //以下三个函数对符号=进行了重载 161. //使String使用=进行一般的赋值 162. //可接受的字符串类型：String,char\*,const char\* 163. /\* 164. String& operator =(String &b){ 165. StrAssign(b); 166. return \*this; 167. } 168. String& operator =(const char\* b){ 169. StrAssign((char\*)b); 170. return \*this; 171. } 172. String& operator =(char\* b){ 173. StrAssign(b); 174. return \*this; 175. } 176. \*/  179. //3、StrConcat函数： 180. //可接受的字符串类型: 181. //String，char\*,const char\* 182. **void** StrConcat(**char**\* b, **int** len){ 183. expand(len + length); 184. **for**(**int** i = 0; i < len; i++) arr[i+length] = b[i]; 185. length += len; 186. arr[length] = 0; 187. } 188. **void** StrConcat(**char**\* b){ 189. StrConcat(b, strlen(b)); 190. } 191. **void** StrConcat(**const** **char**\* b){ 192. StrConcat((**char**\*)b, strlen(b)); 193. } 194. **void** StrConcat(String &b){ 195. StrConcat(b.data(), b.StrLength()); 196. } 198. //4：Substr函数： 199. String Substr(**int** i, **int** j){ 200. String ret; 201. **if**(i < 0) **return** ret; 202. **for**(**int** a = 0; a < j && i + a < length; a++){ 203. ret.push\_back(arr[i + a]); 204. } 205. **return** ret; 206. }  209. //5：StrCmp函数： 210. //可接受的字符串类型：String,char\* 211. //用法： 212. //String a,b;char \* c 213. //a.StrCmp(b),a.StrCmp(c) 214. //比较a与b的大小，比较a与c的大小 215. //a>b则返回1，a=b返回0，a<b返回-1 216. **int** StrCmp(**char**\* b)**const**{ 217. **char**\* a = arr; 218. **while**(\*a && \*b && \*a == \*b) a++, b++; 219. **if**(\*a == 0 && \*b == 0) **return** 0; 220. **if**(\*a && !\*b) **return** 1; 221. **if**(!\*a && \*b) **return** -1; 222. **if**(\*a > \*b) **return** 1; 223. **else** **return** -1; 224. } 225. **int** StrCmp(**const** String &str)**const**{ 226. **return** StrCmp(str.data()); 227. }  230. //6:StrIndex函数： 231. //可接受的类型：String,char\* 232. //用法： 233. //String a,b; 234. //char\* c;const char\* d; 235. //StrIndex(a,b),StrIndex(a,c),StrIndex(a,d) 236. //分别在a中查找字符串b,c,d 237. //如果找到返回第一个下标 238. //如果未找到返回-1 239. **int** StrIndex(**char**\* b, **int** offset){ 240. **char**\* a = arr + offset; 241. **int** index = 0; 242. **while**(\*(a + index)){ 243. **char**\* x = a + index; 244. **char**\* y = b; 245. **while**(\*x && \*y && \*x == \*y) x++, y++; 246. **if**(!(\*y)) **return** index+offset; 247. index++; 248. } 249. **return** -1; 250. } 251. **int** StrIndex(**char**\* b){ 252. **return** StrIndex(b, 0); 253. } 254. **int** StrIndex(**const** **char**\* b){ 255. **return** StrIndex((**char**\*)b, 0); 256. } 257. **int** StrIndex(**const** String& str){ 258. **return** StrIndex(str.data(), 0); 259. } 260. **int** StrIndex(**const** **char**\* b, **int** index){ 261. **return** StrIndex((**char**\*)b, index); 262. } 263. **int** StrIndex(**const** String& str, **int** index){ 264. **return** StrIndex(str.data(), index); 265. } 267. //7、StrInsert函数： 268. //用法: 269. //String a,b;char \* c;int index; 270. //a.StrInsert(b,index),a.StrInsert(c,index); 271. //向a中index位置插入b(c) 272. //同时a中index之后的字符向后移动 273. //如果index<0，index将会赋值为0 274. //如果index大于等于a的大小 275. //那么b(c)会被插入到a的末尾 276. **void** StrInsert(**char**\* str,**int** index,**int** offset){ 277. index = index > length ? length : index; 278. index = index < 0? 0 : index; 279. expand(offset + length); 280. **for**(**int** i = length - 1; i >= index; i--) arr[i + offset] = arr[i]; 281. **for**(**int** i = 0; i < offset; i++) arr[i + index] = str[i]; 282. length += offset; 283. arr[length] = 0; 284. } 285. **void** StrInsert(**char**\* str, **int** index){ 286. StrInsert(str, index, strlen(str)); 287. } 288. **void** StrInsert(**const** String &str, **int** index){ 289. StrInsert(str.data(), index, str.StrLength()); 290. }   294. //8、StrDelete函数: 295. //用法： 296. //String a;int index,int len; 297. //a.StrDelete(index,len) 298. //删除a中由index开始的len个字符 299. **void** StrDelete(**int** index, **int** len){ 300. **if**(index >= length || index < 0) **return**; 301. **if**(index+len >= length){ 302. length = index; 303. arr[length] = 0; 304. **return**; 305. } 306. **for**(**int** i = index + len;i < length; i++) arr[i - len] = arr[i]; 307. length -= len; 308. arr[length] = 0; 309. } 311. //9、StrRep函数: 312. //用法: 313. //String a; 314. //a.StrRep(char\*/String b,char\*/String c) 315. //将a中的b替换为c 316. **void** StrRep(**char**\* a, **char**\* b){ 317. //非递归替换 318. **if**(!\*a) **return**;//如果a是空字符串，直接返回 319. **int** offset = 0; 320. **int** index = StrIndex(a, offset); 321. **int** len\_a = strlen(a); 322. **int** len\_b = strlen(b); 323. **while**(index != -1){ 324. StrDelete(index, len\_a); 325. StrInsert(b, index); 326. index = StrIndex(a, offset); 327. } 328. } 329. **void** StrRep(**const** String &t, **char**\* r){ 330. StrRep(t.data(), r); 331. } 332. **void** StrRep(**char**\* t, **const** String& r){ 333. StrRep(t, r.data()); 334. } 335. **void** StrRep(**const** String &t,**const** String&r){ 336. StrRep(t.data(), r.data()); 337. }  340. }; 341. #include <String.h> 342. //引入String类 343. /\* 344. 题目二 文学研究助手 345. 【问题描述】 346. 文学研究人员经常需要统计某篇英文小说中某些词语出现的次数， 347. 试写一个程序完成该统计要求。 348. 【基本要求】 349. 英文小说存于一个文本文件中（有多行），需要统计的单词由键盘输入。 350. 程序运行结束后输出该关键字在文中出现的总次数 351. 以及出现该关键字的行号和在该行中出现的次数。 352. 【实现提示】 353. 为简化起见，设单词不跨行。 354. 由于文件可能很长，因此，不要试图将文件中所有内容全部读入后才开始统计。 355. 逐行读入文本文件的内容，每读入一行，就在该行中统计一遍指定单词的出现次数。 356. 可以假设每行字符个数不超过120，行号从1开始计数。 357. \*/ 359. /\* 360. 输入格式： 361. 标准输入：第一行n，随后n行，每行一个关键词 362. 之后跟随数行，行数由0开始编号 363. \*/ 364. #define N 10005 365. #define M 200000 366. **int** total\_sum[N];//total\_sum[i]记录第i个关键字出现的总次数 367. **struct** node{ 368. **int** word\_index;//关键字的编号 369. **int** line;//关键字出现的行数 370. **int** sum;//关键字在line行出现的次数 371. }record[M]; 372. **int** record\_cnt;//记录record实际大小 373. **int** main(){ 374. String line(120); 375. **char** l[120]; 376. **char** temp[20]; 377. **int** n; 378. scanf("%d\n",&n); 379. String\* keyword=**new** String[n](); 380. **for**(**int** i=0;i<n;i++){//关键字输入 381. scanf("%[^\n]\n",temp); 382. keyword[i].StrAssign(temp); 383. } 384. **int** line\_num=0; 385. **while**(scanf("%[^\n]\n",l)!=EOF){//读入一行 386. line.StrAssign(l); 387. **for**(**int** i=0;i<n;i++){//对改行检测每个关键字出现次数 388. **int** pos=line.StrIndex(keyword[i]); 389. **while**(pos!=-1){ 390. **if**((pos==0 || line[pos-1]==' ') && (pos+keyword[i].StrLength()==line.StrLength() || line[pos+keyword[i].StrLength()]==' ')){ 391. //表示该单词是一个单词 392. //如关键字是abs，"abc aabc abc" 393. //中间的aabc就不是关键字 394. total\_sum[i]++; 395. record[record\_cnt].sum++; 396. pos=line.StrIndex(keyword[i],pos+keyword[i].StrLength()); 397. }**else**{ 398. pos=line.StrIndex(keyword[i],pos+1); 399. } 400. } 401. **if**(record[record\_cnt].sum!=0){ 402. //记录第i个关键字在第line\_num行出现的次数 403. record[record\_cnt].line=line\_num; 404. record[record\_cnt].word\_index=i; 405. record\_cnt++; 406. } 407. } 408. line\_num++; 409. } 410. sort(record,record+record\_cnt,[&](**const** node&a,**const** node& b){ 411. **if**(a.word\_index!=b.word\_index) **return** a.word\_index<b.word\_index; 412. **else** **return** a.line<b.line; 413. });//对关键字出现记录进行排序 415. **int** j=0; 416. printf("line\tapperance\n"); 417. **for**(**int** i=0;i<n;i++){ 418. **if**(total\_sum[i]==0) **continue**; 419. printf("word:%s\ttotal\_apperance:%d\n",keyword[i].data(),record[i]); 420. **while**(record[j].word\_index==i && j<record\_cnt){ 421. printf("%d\t%d\n",record[j].line,record[j++].sum); 422. } 423. } 424. **for**(**int** i=0;i<n;i++) free(keyword[i].data()); 425. **return** 0; 426. } | | |