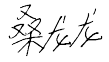
数据结构实验报告

<table>
<tr><td>学号-姓名</td><td>桑龙龙-20030540015</td><td>实验时间</td><td>2020 年 10 月 31 日</td></tr>
<tr><td>诚信声明</td><td colspan="3">本实验及实验报告所写内容为本人所作，没有抄袭。  </td></tr>
<tr><td>实验题目</td><td colspan="3">题目一、字符串基本运算
题目二、文学研究助手</td></tr>
<tr><td>实验过程中遇到的主要问题</td><td colspan="3">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。</td></tr>
<tr><td>实验小结</td><td colspan="3">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</td></tr>
<tr><td>数据结构（自定义数据类型）</td><td colspan="3">字符串的基本操作封装到 String 类中</td></tr>
<tr><td>主要算法（或算法说明）</td><td colspan="3">

```
class String{
    private:
        int length=0;   //字符串实际大小
        int size=0; //字符串拥有空间大小
        char* arr=NULL;//字符串数据存放
        int add = 20;//字符串额外扩容空间
    public:

        //#######构造函数区########
        //构造函数 1：
        //不含任何内容的空字符串
        String(){

        }
        //构造函数 2：
        //将字符串大小初始化为_size
        String(int _size){

            length = _size;
            size = _size+add;
            arr = (char*)malloc(sizeof(char) * size);
            arr[length] = 0;
```

</td></tr>
</table>

```
        }
        //构造函数 3:
        //字符串大小初始化为_size
        //内容都为 val
        String(int _size, char val){

            length = _size;
            size = _size + add;
            arr = (char*)malloc(sizeof(char) * size);
            for(int i=0;i<length;i++) arr[i]=val;
            arr[length] = 0;
        }
        //构造函数 4:
        //将字符串初初始化为 const char*:b
        String(const char* b){
            length = strlen(b);
            size = length + add;
            arr = (char*)malloc(sizeof(char) * size);
            for(int i = 0; i < length; i++) arr[i] = b[i];
            arr[length] = 0;
        }
        //构造函数 5:
        //将字符串初初始化为 char*:b
        String(char* b){
            length = strlen(b);
            size = length+add;
            arr = (char*)malloc(sizeof(char) * size);
            for(int i = 0; i < length; i++) arr[i] = b[i];
            arr[length] = 0;
        }
        //拷贝构造函数:
        //将字符串初初始化为 String:b

        String(const String &b){
            length=b.length;
            size=b.size;
            arr=b.arr;
            add=b.add;
        }

        ~String(){
            //free(arr);
        }

        //######辅助函数区##########
```

```
68.
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.         }
75.
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.        }
120.
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.        }
136.
137.
138.
139.        //#######功能区#########
140.        //1、长度函数:
141.        int StrLength()const{//1
142.            return length;
143.        }
144.
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.         */
177.
178.
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.         }
197.
198.         //4：Substr 函数：
199.         String Substr(int i, int j){
200.             String ret;
201.             if(i < 0) return ret;
```

```
            for(int a = 0; a < j && i + a < length; a++){
                ret.push_back(arr[i + a]);
            }
            return ret;
        }


        //5：StrCmp 函数：
        //可接受的字符串类型：String,char*
        //用法：
        //String a,b;char * c
        //a.StrCmp(b),a.StrCmp(c)
        //比较 a 与 b 的大小，比较 a 与 c 的大小
        //a>b 则返回 1，a=b 返回 0，a<b 返回-1
        int StrCmp(char* b)const{
            char* a = arr;
            while(*a && *b && *a == *b) a++, b++;
            if(*a == 0 && *b == 0) return 0;
            if(*a && !*b) return 1;
            if(!*a && *b) return -1;
            if(*a > *b) return 1;
            else return -1;
        }
        int StrCmp(const String &str)const{
            return StrCmp(str.data());
        }


        //6:StrIndex 函数：
        //可接受的类型：String,char*
        //用法：
        //String a,b;
        //char* c;const char* d;
        //StrIndex(a,b),StrIndex(a,c),StrIndex(a,d)
        //分别在 a 中查找字符串 b,c,d
        //如果找到返回第一个下标
        //如果未找到返回-1
        int StrIndex(char* b, int offset){
            char* a = arr + offset;
            int index = 0;
            while(*(a + index)){
                char* x = a + index;
                char* y = b;
                while(*x && *y && *x == *y) x++, y++;
                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.         }
266.
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.         }
291.
292.
293.
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.         }
310.
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.         }
338.
339.
340. };
```

```
#include <String.h>
//引入 String 类
/*
题目二 文学研究助手
【问题描述】
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，
试写一个程序完成该统计要求。
【基本要求】
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。
程序运行结束后输出该关键字在文中出现的总次数
以及出现该关键字的行号和在该行中出现的次数。
【实现提示】
为简化起见，设单词不跨行。
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。
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。
可以假设每行字符个数不超过 120，行号从 1 开始计数。
*/

/*
输入格式：
标准输入：第一行 n，随后 n 行，每行一个关键词
之后跟随数行，行数由 0 开始编号
*/
#define N 10005
#define M 200000
int total_sum[N];//total_sum[i]记录第 i 个关键字出现的总次数
struct node{
    int word_index;//关键字的编号
    int line;//关键字出现的行数
    int sum;//关键字在 line 行出现的次数
}record[M];
int record_cnt;//记录 record 实际大小
int main(){
    String line(120);
    char l[120];
    char temp[20];
    int n;
    scanf("%d\n",&n);
    String* keyword=new String[n]();
    for(int i=0;i<n;i++){//关键字输入
        scanf("%[^\n]\n",temp);
        keyword[i].StrAssign(temp);
    }
```

```
    int line_num=0;
    while(scanf("%[^\n]\n",l)!=EOF){//读入一行
        line.StrAssign(l);
        for(int i=0;i<n;i++){//对改行检测每个关键字出现次数
            int pos=line.StrIndex(keyword[i]);
            while(pos!=-1){
                if((pos==0 || line[pos-1]==' ') && (pos+keyword[i].StrLength()==line.StrLength() || line[pos+keyword[i].StrLength()]==' ')){
                    //表示该单词是一个单词
                    //如关键字是 abs，"abc aabc abc"
                    //中间的 aabc 就不是关键字
                    total_sum[i]++;
                    record[record_cnt].sum++;
                    pos=line.StrIndex(keyword[i],pos+keyword[i].StrLength());
                }else{
                    pos=line.StrIndex(keyword[i],pos+1);
                }
            }
            if(record[record_cnt].sum!=0){
                //记录第 i 个关键字在第 line_num 行出现的次数
                record[record_cnt].line=line_num;
                record[record_cnt].word_index=i;
                record_cnt++;
            }
        }
        line_num++;
    }
    sort(record,record+record_cnt,[&](const node&a,const node& b){
        if(a.word_index!=b.word_index) return a.word_index<b.word_index;
        else return a.line<b.line;
    });//对关键字出现记录进行排序

    int j=0;
    printf("line\tapperance\n");
    for(int i=0;i<n;i++){
        if(total_sum[i]==0) continue;
        printf("word:%s\ttotal_apperance:%d\n",keyword[i].data(),record[i]);
        while(record[j].word_index==i && j<record_cnt){
            printf("%d\t%d\n",record[j].line,record[j++].sum);
```

```
        }
    }
    for(int i=0;i<n;i++) free(keyword[i].data());
    return 0;
}
```