补充：

5、 TString 类的定义与实现

5.1. 引入

字符串的基本操作都是基于一维字符数组的，因此其赋值、比较、连接、求长度等方法均要用函数去实现，且实现过程中必须注意空间是否足够、最后是不是有'\0'等情况。

5.2. 实现目标

参照 string 类的方法，用比较简洁易懂的方法来实现字符串的基本操作。

5.3. 要求实现的基本操作

5.3.1. 定义对象并初始化：

```
① TString s1;                    //s1 为空
② TString s1("hello");           //s1 为"Hello"
③ TString s1="hello";            //s1 为"Hello"
④ TString s1("Hello"), s2=s1;    //s1/s2 均为"Hello"
⑤ char *s = "Hello";
   TString s1 = s;               //s1 为"Hello"
⑥ char s[] = "Hello";
   TString s1 = s;               //s1 为"Hello"
```

5.3.2. 输入操作：（以空格/回车做为输入结束）

```
① TString s1;
   cin >> s1;   //若键盘输入 Hello,则 s1 得到"Hello"
② TString s1;
   cin >> s1;   //若键盘输入 Hello 123,则 s1 得到"Hello"(空格为分隔符)
```

5.3.3. 输出操作：

```
① TString s1("hello");
   cout << s1;   //输出"hello"
② TString s1;
   cout << s1;   //输出"<empty>"，说明：这是对 NULL 进行的特殊处理
```

5.3.4. 取字符串操作：（将 TString 中的内容以 char *方式返回，只读，不可更改内容）

```
① TSTring s1("hello");
   printf("%s", s1.c_str());     //输出为"hello"
```

5.3.5. 赋值操作：

```
① TString s1("hello"), s2;
   s2=s1;         //s2 为"Hello"
② TString s1("Hello");
   s1="Hi";       //s1 为"Hi",原"Hello"不再保留
```

5.3.6. 连接操作：（运算符+表示字符串连接后赋值给另一个串）

```
① TString s1("tong"), s2("ji"), s3;
   s3 = s1+s2;     //s3 为"tongji"
   s3 = s2+s1;     //s3 为"jitong"
② TString s1("tong"), s3;
   s3 = s1+"ji";   //s3 为"tongji"
   s3 = "ji"+s1;   //s3 为"jitong"
③ TString s1("tong"), s3;
   char *s="ji";
   s3 = s1+s;      //s3 为"tongji"
   s3 = s+s1;      //s3 为"jitong"
```

④ TString s1("tong"), s3;
```
char s[]="ji";
s3 = s1+s;      //s3为"tongji"
s3 = s+s1;      //s3为"jitong"
```
⑤ TString s1("Hello"), s3
```
char c = '!';
s3 = s1 + c;              //s3为"Hello!"
```
⑥ TString s1("ello"), s3
```
s3 = 'H' + s1;            //s3为"Hello"
```
5.3.7. 自连接操作：（运算符+=表示字符串连接后赋值给自己）

① TString s1("tong"), s2("ji");
```
s1 += s2;       //s1为"tongji"
```
② TString s1("tong");
```
s1 += "ji";     //s1为"tongji"
```
③ TString s1("tong");
```
char *s="ji";
s1 += s;        //s1为"tongji"
```
④ TString s1("tong");
```
char s[]="ji";
s1 += s;        //s1为"tongji"
```
⑤ TString s1("Hello");
```
char c = '!';
s1 += c;        //s1为"Hello!"
```
5.3.8. 自连接操作的等价操作 append()

⑥ TString s1("tong"), s2("ji");
```
s1.append(s2);  //s1为"tongji"
```
⑦ TString s1("tong");
```
s1.append("ji");//s1为"tongji"
```
⑧ TString s1("tong");
```
char *s="ji";
s1.append(s);   //s1为"tongji"
```
⑨ TString s1("tong");
```
char s[]="ji";
s1.append(s);   //s1为"tongji"
```
⑩ TString s1("Hello");
```
char c = '!';
s1.append©;     //s1为"Hello!"
```
5.3.9. 删除操作：（运算符-表示从字符串中删除另一个字符串/一个字符后赋值给另一个串）

① TString s1("tongji"), s2("ji"), s3;
```
s3 = s1 - s2;         //s3为"tong"
```
② TString s1("tongji"), s3;
```
s3 = s1 - "ji";       //s3为"tong"
```
③ TString s1("tongji"), s3;
```
char *s="ji";
s3 = s1 - s;          //s3为"tong"
```
④ TString s1("tongji"), s3;
```
char s[]="ji";
s3 = s1 - s;          //s3为"tong"
```

⑤ TString s1("tongji"), s3;
char c1 = 'j', c2 = 'i';
s3 = s1 - c1 - c2; //s3 为"tong"

5.3.10. 自删除操作：（运算符-=表示从字符串中删除另一个字符串/一个字符后赋值给自己）

① TString s1("tongji"), s2("ji");
s1 -= s2; //s1 为"tong"

② TString s1("tongji");
s1 -= "ji"; //s1 为"tong"

③ TString s1("tongji");
char *s="ji";
s1 -= s; //s1 为"tong"

④ TString s1("tongji");
char s[]="ji";
s1 -= s; //s1 为"tong"

⑤ TString s1("tongji");
char c1 = 'j', c2 = 'i';
(s1 -= c1) -= c2; //s1 为"tong"

5.3.11. 复制操作：（运算符*表示将字符串自身复制若干倍后赋值给另一个串）

① TString s1("tong"), s2;
s2 = s1*2; //s2 为"tongtong"

② TString s1, s2;
s2 = s1*5; //s2 为<NULL> "

5.3.12. 自复制操作：（运算符*表示将字符串自身复制若干倍后赋值给自己）

① TString s1("tong"), s2;
s1 *= 2; //s1 为"tongtong"

② TString s1;
s1 *= 5; //s1 为<NULL> "

5.3.13. 反转操作：（运算符!表示将字符串反转后赋值给另一个串）

① TString s1("tong"), s2;
s2 = !s1; //s2 为"gnot"，s1 仍为"tong"

② TString s1;
s2 = !s1; //s2 为<NULL> "

5.3.14. 比较操作：（按 strcmp 的规则返回即可）

① TString s1="house", s2="horse";
s1 > s2; (包括其它 5 种比较运算) //返回 0/1

② TString s1="house";
s1 > "horse"; (包括其它 5 种比较运算) //返回 0/1
"horse" > s2; (包括其它 5 种比较运算) //返回 0/1

③ TString s1="house";
char *s="horse"
s1 > s; (包括其它 5 种比较运算) //返回 0/1
s > s2; (包括其它 5 种比较运算) //返回 0/1

④ TString s1="house";
char s[]="horse"
s1 > s; (包括其它 5 种比较运算) //返回 0/1
s > s2; (包括其它 5 种比较运算) //返回 0/1

5.3.15. 求串长度：（按 strlen 的规则返回即可）

① TString s1("Hello");
cout << s1.length(); //输出为 5

② 定义全局函数 TStringLen(const TString &);

```
TString s1("Hello"), s2("123");
char *s3="abcde";
char s4[]="wxyz";
TStringLen(s1+s2);     //返回值为 8
TStringLen(s2+s1);     //返回值为 8
TStringLen(s1+"pq");   //返回值为 7
TStringLen("pq"+s1);   //返回值为 7
TStringLen(s1+s3);     //返回值为 10
TStringLen(s3+s1);     //返回值为 10
TStringLen(s1+s4);     //返回值为 9
TStringLen(s4+s1);     //返回值为 9
```

5.3.16. 取串中某个字符的值/给串中的某个字符赋值：（按字符数组的规则即可）

① TString s1("hello");

```
cout << s1[1];          //输出为 e
```

② TString s1("hello");

```
s1[0] -= 32;
cout << s1;             //输出为 Hello
```

【要求:】1、程序由三个文件组成，各文件的说明如下：

16-b5.h：给出 TString 类的定义及其它需要的定义

16-b5.cpp：给出 TString 类的所有成员函数的实现及其它需要的全局函数的实现

16-b5-main.cpp：在 main 函数中给出了 TString 类的测试用例，不准修改，不需要提交，检查作业时会替换本文件

2、在操作系统的内存允许的情况下，**均不再考虑空间是否够用，但也不能浪费空间**

例 1：TString s1("Hello"); 则最多允许申请 6 个字节的空间，**不能多申请**

★ 也可以申请 5 个字节，即不用\0 表示结束，直接用 len 表示长度（不强求）

例 2：TString s1; 若 s1 = ***** 或 s1 = s1 + ***** 等语句反复出现时，不能简单限定空间不超过多少字节，**要无尽利用空间，直到内存空间被耗尽为止**

例 3：TString s1("Hello"); 若 s1 = s1-"He";则 s1 占用空间要减为最多 4 字节

5、实现过程**不允许**使用系统提供的 string 类，但可以使用<cstring>中字符串函数

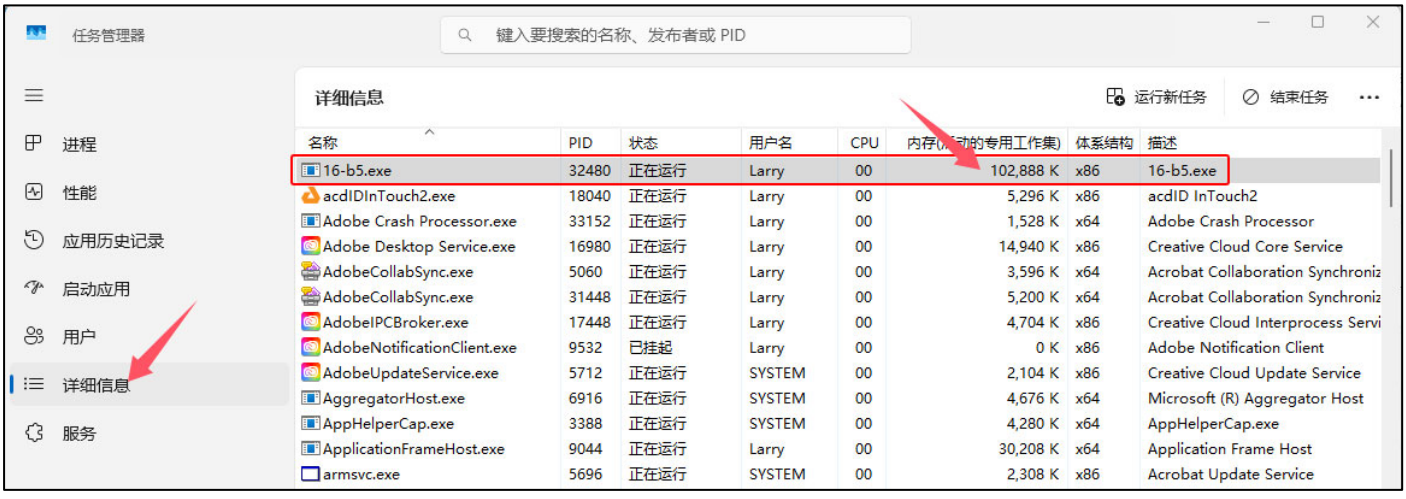
6、最后的 100MB 累加测试完成后，用“任务管理器”查看的内存占用**不能超过 115MB**

7、最后的 100MB 累加测试，+=方式的完成时间，**不允许**超过+方式的 60%；+=的两种方式的时间应该一样（考虑到实际测试环境，允许±5%）

8、给出 Windows 下的 16-b5-demo.exe 供参考

9、给出 Linux 下的 16-b5-demo 供参考（在$下输入 16-b5-demo 即可运行，不需要./）

10、**Linux 限定**每个用户 shell 的最大可用内存为 408MB，必须在此限制内完成 100MB 累加



【提示：】

1、笔记本插电/不插电的性能会有差异，具体看机器的设置

2、100MB测试的结果，对时间差异影响最大的应该是内存条的速度，DDR5应该能碾压DDR4

3、大家可以把100MB的速度贴在群里，相互验证一下机器的性能

```
s1已有长度：     99.974 MB字节，本次增加 37842 字节
s1已有长度：    100.025 MB字节，本次增加 53301 字节，总用时    168.625秒，本次1MB用时      3.297秒
time=168.641
内存分配到达满100MB，测试结束
    本次测试耗时 168.641秒
内存性能测试(s1=s1+str方式)
    老师的机器（CPU：Intel i7-9700F , 内存：DDR4 2666MHz）运行VS2022-Debug-x86编译的程序，大约耗时150-180秒
    【说明】：只有相同编译器下的运行时间才有可比性
              如果时间相差太大，除CPU和内存的性能差异外，还有可能是算法问题

观察任务管理器中本程序的内存占用情况(不允许超过115MB)...
```

```
s1已有长度：     99.916 MB字节，本次增加 38329 字节
s1已有长度：    99.9485 MB字节，本次增加 34158 字节
s1已有长度：    100.004 MB字节，本次增加 58215 字节，总用时    138.268秒，本次1MB用时       2.65秒
time=138.268
内存分配到达满100MB，测试结束
    本次测试耗时 138.268秒
内存性能测试(s1=s1+str方式)
    Linux服务器下运行，大约耗时135-155秒（如果多人同时测试，偏差可能较大）
    【说明】：只有相同编译器下的运行时间才有可比性
              如果时间相差太大，除CPU和内存的性能差异外，还有可能是算法问题

按回车键继续
```

```
s1已有长度：    99.9637 MB字节，本次增加 33049 字节
s1已有长度：    99.9976 MB字节，本次增加 35572 字节
s1已有长度：    100.043 MB字节，本次增加 47580 字节，总用时     85.406秒，本次1MB用时      1.656秒
time=85.422
内存分配到达满100MB，测试结束
    本次测试耗时 85.422秒
内存性能测试(s1 += str方式)
    老师的机器（CPU：Intel i7-9400F , 内存：DDR4 2666MHz）运行VS2022-Debug-x86编译的程序，大约耗时80-100秒
    【说明】：只有相同编译器下的运行时间才有可比性
              如果时间相差太大，除CPU和内存的性能差异外，还有可能是算法问题

观察任务管理器中本程序的内存占用情况(不允许超过115MB)...
```

```
s1已有长度：    99.9291 MB字节，本次增加 35339 字节
s1已有长度：    99.9798 MB字节，本次增加 53190 字节
s1已有长度：    100.026 MB字节，本次增加 48475 字节，总用时     80.634秒，本次1MB用时       1.39秒
time=80.634
内存分配到达满100MB，测试结束
    本次测试耗时 80.634秒
内存性能测试(s1 += str方式)
    Linux服务器下运行，大约耗时70-90秒（如果多人同时测试，偏差可能较大）
    【说明】：只有相同编译器下的运行时间才有可比性
              如果时间相差太大，除CPU和内存的性能差异外，还有可能是算法问题

按回车键继续
```

```
s1已有长度：    99.9304 MB字节，本次增加 60131 字节
s1已有长度：    99.9682 MB字节，本次增加 39561 字节
s1已有长度：    100.003 MB字节，本次增加 36288 字节，总用时     84.875秒，本次1MB用时      1.485秒
time=84.89
内存分配到达满100MB，测试结束
    本次测试耗时 84.89秒
内存性能测试(s1 += s2方式)
    老师的机器（CPU：Intel i7-9400F , 内存：DDR4 2666MHz）运行VS2022-Debug-x86编译的程序，大约耗时80-100秒
    【说明】：只有相同编译器下的运行时间才有可比性
              如果时间相差太大，除CPU和内存的性能差异外，还有可能是算法问题

观察任务管理器中本程序的内存占用情况(不允许超过115MB)...
```

```
s1已有长度:    99.9215 MB字节，本次增加 55609 字节
s1已有长度:    99.9589 MB字节，本次增加 39239 字节
s1已有长度:    100.015 MB字节，本次增加 59065 字节，总用时    79.362秒，本次1MB用时       1.4秒
time=79.362
内存分配到达满100MB，测试结束
    本次测试耗时 79.362秒
内存性能测试(s1 += s2方式)
    Linux服务器下运行，大约耗时70-90秒（如果多人同时测试，偏差可能较大）
    【说明】：只有相同编译器下的运行时间才有可比性
              如果时间相差太大，除CPU和内存的性能差异外，还有可能是算法问题

按回车键继续
```

6、 enum class 的重载

6.1. 引入

由第 14 模块知识可知，enum 的常量/变量直接用 cout 输出得到的是 int 型，而 enum class 的常量/变量是无法用 cout 直接输出的，直接原因就是 enum 及 enum class 为用户自定义类型，因此 istream/ostream 无法直接匹配，而解决方法就是运算符重载

```
#include <iostream>
using namespace std;
enum week { sun, mon, tue, wed, thu, fri, sat };
int main()
{
    cout << sun << ' ' << mon << endl;  //正确，输出 int
    printf("%d %d\n", sun, mon);        //正确，输出 int
    return 0;
}
```

```
#include <iostream>
using namespace std;
enum class week { sun, mon, tue, wed, thu, fri, sat };
int main()
{
    cout << week::wed << endl;       //本句报错
error C2679: 二元“<<”: 没有找到接受“week”类型的右操作数的运算符(或没有可接受的转换)
    cout << int(week::wed) << endl; //正确，输出 int
    printf("%d\n", week::mon);      //正确，输出 int
    return 0;
}
```

6.2. 实现目标

用运算符重载实现 enum class 的 cin/cout 重载及部分符合语义要求的运算。

6.3. 要求实现的基本操作

已知 enum week { sun, mon, tue, wed, thu, fri, sat }，要求实现下面的操作

6.3.1. cout 方式直接输出 week 型常量/变量，能得到"星期 x"

6.3.2. cin 方式直接输入 week 型变量，在输入为"sun"(大小写不敏感)时输出为"星期日"

6.3.3. ++/--操作，使变量值在 sun~sat 循环往复

6.3.4. +/- n 操作，在 n 允许任意值的情况下，使变量值在 sun~sat 循环往复

6.3.5. +=/-= n 操作，在 n 允许任意值的情况下，使变量值在 sun~sat 循环往复

【要求：】1、程序由三个文件组成，各文件的说明如下：
16-b6.h：给出 enum class 的定义及其它需要的定义
16-b6.cpp：给出 16-b6.h 中定义的具体实现
16-b6-main.cpp：在 main 函数中给出了 enum class 的测试用例，不准修改，不需要提交，检查作业时会替换本文件
2、注意：如果 16-b6.cpp 的实现在 Windows/Linux 有所不同，用条件编译

7、 enum 的重载
【要求：】1、基本要求：同 16-b6，将 enum class week 换为 enum 即可
2、程序由三个文件组成，各文件的说明如下：
16-b7.h：给出 enum class 的定义及其它需要的定义
16-b7.cpp：给出 16-b7.h 中定义的具体实现
16-b7-main.cpp：在 main 函数中给出了 enum 的测试用例，不准修改，不需要提交，检查作业时会替换本文件
3、将 16-b6-main.cpp 的内容全部复制得到 16-b7-main.cpp 中，也需要测试全部通过
4、注意：如果 16-b6.cpp 的实现在 Windows/Linux 有所不同，用条件编译

【编译器要求：】

		编译器VS	编译器Dev	编译器Linux
16-b5.h	TString类-头文件	Y	Y	Y
16-b5.cpp	TString类-实现	Y	Y	Y
16-b6.h	enumclass重载-头文件	Y	Y	Y
16-b6.cpp	enumclass重载-实现	Y	Y	Y
16-b7.h	enum重载-头文件	Y	Y	Y
16-b7.cpp	enum重载-实现	Y	Y	Y

【作业要求：】
1、12 月 5 日前网上提交本次作业
2、每题所占平时成绩的具体分值见网页
3、超过截止时间提交作业则不得分