

实验二 类/对象

1.实验目的要求

- (1)掌握类的定义和实现。
- (2)掌握对象创建及使用的基本方法。
- (3)掌握全局变量和局部变量、动态变量和静态变量的概念和使用方法。

2.实验内容

<1> 根据全局变量和局部变量的特性，分析程序的输出结果。

```
1  #include <iostream>
2  using namespace std;
3
4  int k = 1;
5
6  int main()
7  {   int i = 4 ;
8      void fun (int);
9      fun( i ) ;
10     cout << "(1)" << i << ',' << k << endl ;
11     return 0;
12 }
13
14 void fun( int m )
15 {
16     m += k;
17     k += m;
18     {
19         char k = 'B';
20         cout << "(2)" << char(k-1) << endl ;
21     }
22     cout << "(3)" << m << ',' << k << endl ;
23 }
```

输出结果：

```
1  (2)A
2  (3)5,6
3  (1)4,6
```

结果分析:

对于第一行输出结果 (2)A 由于 char(k-1) 中使用的k变量并不是我们当初设定好的全局变量 int k = 1，而是程序第十九行设置的局部变量 char k = 'B'，遵循局部变量优先于全局变量，所以 char(k-1) 中的k是字符B对应的ASCII码所对应的整数值，将其减一并再将数据类型转化为char类型，得到的就是ASCII码表中B前面的一个单位A

对于第二行输出结果，此时用到的k就是在程序当初定义的全局变量k，将参数i=4传入函数中作为m变量，即m的值起初为4，k的值为1，经过 `m+=k` 语句之后，m的值为5，k的值为1。接下来执行语句 `k+=m`，此时k的值为6，m的值为5，故这一行输出为 (3)5,6

对于第三行输出结果，由于在上一行中对fun函数的调用并没有传递地址，为值传递，不会改变i变量的值，故i变量的值仍旧为4，而对于k，fun函数中对其的操作可以改变其变量本身的值，故k的值仍旧为第二行操作之后的值为6，故一行输出为 (1)4,6

<2> 根据全局变量、局部变量和静态变量的特性，分析运行输出结果。问题(1)处语句有没有static，运行结果会发生改变么？为什么？

```
1  #include <iostream>
2  #include<iomanip>
3  using namespace std;
4
5  void subp ( )
6  {   static int x = 0 ,   y = 0 ; //问题 (1)
7      int a = 1, b = 1;
8      a = a + x ;
9      b = b + y ;
10     cout<<"subp函数输出: \n";
11     cout << setw(5) << a << setw(5) << b << '\n' ;
12     cout << setw(5) << x << setw(5) << y << '\n' ;
13     x++;
14     y++;
15 }
16
17 int x,y;
18
19 int main()
20 {   int a = 9, b = 3;
21     x = a - b ;
22     y = a + b ;
23     subp();
24     cout<<"main函数输出: \n";
25     cout << setw(5) << a << setw(5) << b << '\n' ;
26     cout << setw(5) << x << setw(5) << y << '\n' ;
27     subp();
28     return 0;
29 }
```

输出结果如下：

```
1  subp函数输出:
2      1      1
3      0      0
4  main函数输出:
5      9      3
6      6     12
7  subp函数输出:
8      2      2
9      1      1
```

思考题解答：

- 对输出结果进行分析：

对于第一部分subp函数输出结果，函数中x y变量的值为函数内部定义的静态局部变量的值，都为0，且ab两个变量都加上xy这两个为0的变量，故输出为两个1和两个0

对于第二部分main函数的输出，由于subp中定义的静态局部变量只在定义它的函数中可见，故main函数中xy这两个变量并不是subp函数中的那两个，而是分别是 a-b, a+b 的值，即为6和12。同时由于此题中函数内部定义的局部变量的作用域仅仅在定义其的函数当中，故main函数中ab的值为main函数当中定义ab值，即为9和3

对于第三部分subp函数的输出，由于静态局部变量所在的函数在多调用多次时，只有第一次才经历变量定义和初始化，以后多次在调用时不再定义和初始化，而是维持之前上一次调用时执行后这个变量的值。故xy的初始值为上一次调用subp函数的结果，则ab各加上1，则ab各为2，xy为上次xy的结果值，都为1

<3>设计了成员函数将两个Time对象相加（即时间相加），并进行相应的检查，查看增加的分钟数及秒数是否大于59。如果秒数大于59，则分钟数向前递增1。类似地，如果分钟数大于59，则小时数向前增1。

```
1  #include <iostream>
2  using namespace std;
3
4  class Time{
5  private:
6      int hours, minutes, seconds;
7  public:
8      void set_time(){
9          cin>>hours>>minutes>>seconds;
10     }
11     void display_time(){
12         cout<<hours<<':'<<minutes<<':'<<seconds<<endl;
13     }
14     void add_time(Time & t1, Time & t2){
15         hours=t1.hours+t2.hours;
16         minutes=t1.minutes+t2.minutes;
17         seconds=t1.seconds+t2.seconds;
18         if(seconds>=60){
19             seconds-=60;
20             minutes++;
21         }
22         if(minutes>=60){
23             minutes-=60;
24             hours++;
25         }
26     }
27 };
28
29 int main()
30 {
31     Time one, two, three;
32     cout<<"\nEnter the first time(hours minutes seconds):";
33     one.set_time();
34     cout<<"\nEnter the second time(hours minutes seconds):";
35     two.set_time();
36     three.add_time(one,two);
37     cout<<"the result is:"<<endl;
38     three.display_time();
```

```
39     return 0;
40 }
41
```

基本要求：

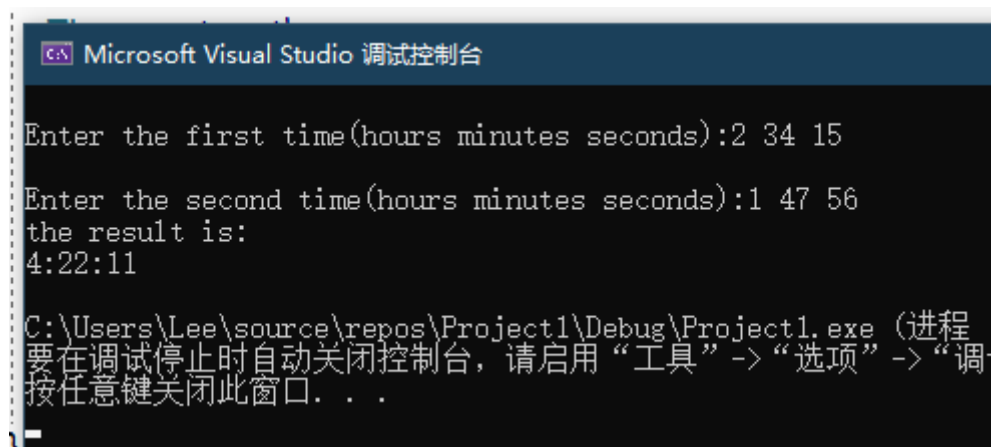
- 上机录入，调试上面程序
- 运行程序，输入下面两组数据
 - ① 2 34 15
1 47 56
 - ② 2 67 100
1 56 200

分析运行结果是否正确

分析与思考：

- 1) 增加构造函数对Time类的对象进行初始化。
- 2) 该程序要求用户输入的分钟数和秒数必须小于60，如何修改程序使得用户在输入分钟数和秒数大于等于60时，也能得到正确的结果。

第一组的数据的输出结果为：



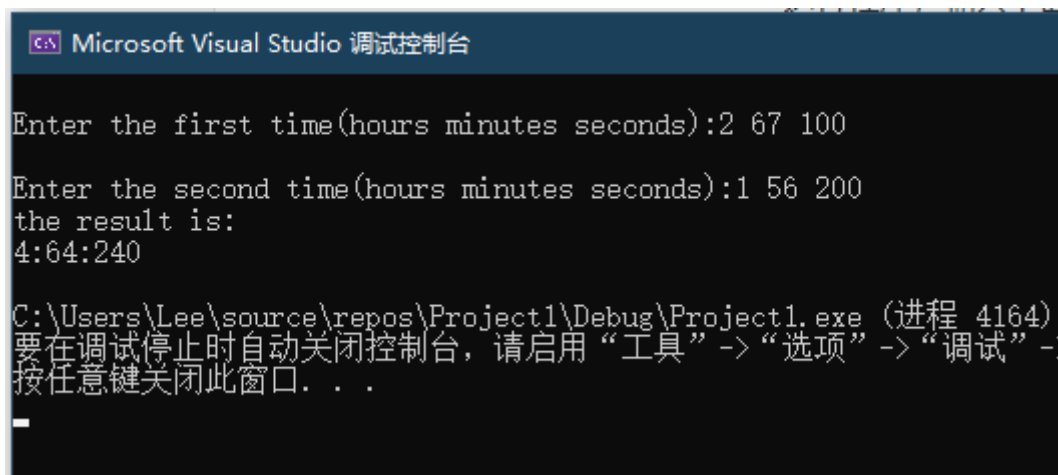
```
Microsoft Visual Studio 调试控制台

Enter the first time(hours minutes seconds):2 34 15
Enter the second time(hours minutes seconds):1 47 56
the result is:
4:22:11

C:\Users\Lee\source\repos\Project1\Debug\Project1.exe (进程
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调
按任意键关闭此窗口. . .
```

经过分析计算可知，第一组数据的结果是正确的。15+56得到71，根据秒数每满60进一分钟的原则，减去60得到11，同时分钟数+1，而34+47得到81，再加上之前秒数进位的1，得到82，根据分钟数每满60进一小时的原则，减去60得到22，并且小时数进一，对于小时部分，2+1得到3，再加上之前分钟数进位的1，得到4，故最终结果为 4:22:11 符合程序输出结果，是正确的

第二组数据的输出结果为：



```
Microsoft Visual Studio 调试控制台

Enter the first time(hours minutes seconds):2 67 100

Enter the second time(hours minutes seconds):1 56 200
the result is:
4:64:240

C:\Users\Lee\source\repos\Project1\Debug\Project1.exe (进程 4164) 已
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->
按任意键关闭此窗口. . .
```

显然第二组数据是错误的, 它的分钟数和秒数都大于60了, 错误原因就在于源代码中, 对于大于60的, 无论有多大, 都只进一次位, 故造成了我们意料之外的结果。

分析与思考解答:

1) 添加代码如下:

```
1 public:
2     Time()
3     {
4         hours = 0;
5         minutes = 0;
6         seconds = 0;
7     }
```

2) 代码修改如下:

由于是if语句中的问题, 故程序的其他部分我们不做改动, 仅仅修改if语句部分, 将需要进位的部分, 除以进位的每个单位制(都为60), 得到需要进位的次数, 即可解决第二组数据中出现的问

```
1 if (seconds >= 60) {
2     int flag = seconds / 60;
3     seconds -= 60*flag;
4     minutes += flag;
5 }
6 if (minutes >= 60) {
7     int flag = minutes / 60;
8     minutes -= 60*flag;
9     hours+=flag;
10 }
```

改进后程序的运行结果:

结束。

```
选择Microsoft Visual Studio 调试控制台

Enter the first time(hours minutes seconds):2 67 100
Enter the second time(hours minutes seconds):1 56 200
the result is:
5:8:0

C:\Users\Lee\source\repos\Project1\Debug\Project1.exe (
要在调试停止时自动关闭控制台，请启用“工具”->“选项”-
按任意键关闭此窗口. . .
```

最后修改过后的程序源代码如下：

```
1  #include <iostream>
2  using namespace std;
3
4  class Time {
5  private:
6      int hours, minutes, seconds;
7  public:
8      Time()
9      {
10         hours = 0;
11         minutes = 0;
12         seconds = 0;
13     }
14  public:
15     void set_time() {
16         cin >> hours >> minutes >> seconds;
17     }
18     void display_time() {
19         cout << hours << ':' << minutes << ':' << seconds << endl;
20     }
21     void add_time(Time& t1, Time& t2) {
22         hours = t1.hours + t2.hours;
23         minutes = t1.minutes + t2.minutes;
24         seconds = t1.seconds + t2.seconds;
25         if (seconds >= 60) {
26             int flag = seconds / 60;
27             seconds -= 60*flag;
28             minutes += flag;
29         }
30         if (minutes >= 60) {
31             int flag = minutes / 60;
32             minutes -= 60*flag;
33             hours+=flag;
34         }
35     }
36 };
37
38 int main()
39 {
40     Time one, two, three;
41     cout << "\nEnter the first time(hours minutes seconds):";
42     one.set_time();
```

```

43     cout << "\nEnter the second time(hours minutes seconds):";
44     two.set_time();
45     three.add_time(one, two);
46     cout << "the result is:" << endl;
47     three.display_time();
48     return 0;
49 }

```

<4> 阅读下面的一段程序代码，代码可能有错误，请仔细分析并体会。

```

1  #include <iostream>
2  using namespace std;
3
4  class Date {
5      public:
6          static bool IsLeapyear;
7          Date(){};
8          Date(int year,int month,int day);
9          ~Date(){};
10         int &GetYear(){return year;}
11         int &GetMonth(){return month;}
12         int &GetDay(){return day;}
13     private:
14         int year;
15         int month;
16         int day;
17 };
18
19 bool Date::IsLeapyear=true;
20
21 int Date::Date(int year,int month,int day)
22 {
23     (*this).year=year;
24     (*this).month=month;
25     (*this).day=day;
26 }
27
28 int main()
29 {
30     int year,month,day;
31     cin>>year>>month>>day;
32     Date mydate(year,month,day);
33     int &myyear=mydate.GetYear();
34     int &mymonth=mydate.GetMonth();
35     int &myday=mydate.GetDay();
36     cout<<myyear<<"-"<<mymonth<<"-"<<myday<<'\\n';
37     cout<<mydate.GetYear()<<"-"<<mydate.GetMonth()<<"-"<<mydate.GetDay()
<<'\\n';
38     myyear = 2003;
39     mymonth = 9;
40     myday = 1;
41     cout<<mydate.GetYear()<<"-"<<mydate.GetMonth()<<"-"<<mydate.GetDay()
<<'\\n';
42     return 0;
43 }

```

仔细观察程序，会发现如下错误：

1. 第六行和第八行 `void Date(){};` 和 `void ~Date(){};`

我们知道c++的构造函数和析构函数是无返回值类型的，函数前也不需要加void，故改进方法是把void删掉

改进后的代码如下：

```
1 Date(){};
2 ~Date(){};
```

2. 第七行的带参构造函数前同样不能带返回值类型，且在类外初始化这个函数的时候，也不带返回值类型

错误代码：

```
1 int Date(int year,int month,int day);
```

```
1 int Date::Date(int year,int month,int day)
2 {
3     (*this).year=year;
4     (*this).month=month;
5     (*this).day=day;
6 }
```

改进方法如下：

去掉两个 `int`

改进后如下：

```
1 Date(int year,int month,int day);
```

```
1 Date::Date(int year,int month,int day)
2 {
3     (*this).year=year;
4     (*this).month=month;
5     (*this).day=day;
6 }
```

- 3.对程序第18行中 `bool Date::IsLeapyear=true;`，这个静态变量没有在类内声明，就在类外初始化了，这在c++当中是不允许的

改进如下：

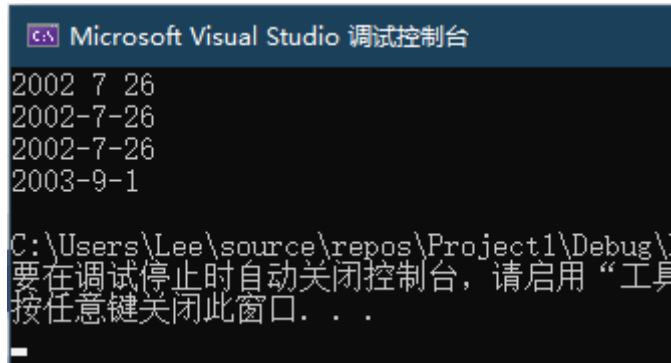
```
1 class Date {
2     public:
3         static bool IsLeapyear;//添加的代码
4         Date(){};
5         Date(int year,int month,int day);
6         ~Date(){};
7         int &GetYear(){return year;}
8         int &GetMonth(){return month;}
9         int &GetDay(){return day;}
10    private:
11        int year;
```



```
12     int month;
13     int day;
14 };
```

4.对于程序第32行, 类对象实例化存在问题 `Date mydate(year,month,day);`
应改进为 `Date mydate = Date (year, month, day);`

运行得到输出结果如下:



分析与思考:

main函数中`int &myyear=mydate.GetYear();`、`int &mymonth=mydate.GetMonth();`和`int &myday=mydate.GetDay();`这些语句表达是**引用的思想**, 通过返回值类型为引用的变量, 来给类中的原本的成员变量起"别名", 同时能给通过对引用类型变量的操作, 也能改变成员变量的值, 即引用类型的那个变量和原来的变量, 指向的是同一块内存。

在本例中, 这种方法是不好的, 通过观察本例子中Date类的结构, 它的三个变量都是通过private修饰符修饰了, 表明这三个变量它并不希望它的初始化之后, 有外部力量来改变它, 而我们这里若通过引用类型的变量做返回值了, 例如我们对myyear变量的任何操作, 也能造成Date内private修饰的year变量的值的变化, 这并不是当初设计这个类的人所期望看到的。

应该把`GetYear()`, `GetMonth()`, `GetDay()`三个函数的返回值设置成int, 而不是int &

相关代码改进如下:

```
1  int GetYear(){return year;}
2  int GetMonth(){return month;}
3  int GetDay(){return day;}
4
5  int myyear=mydate.GetYear();
6  int mymonth=mydate.GetMonth();
7  int myday=mydate.GetDay();
```