一份后交的实验四作业

——完美演绎上周总结的错误。

```
char alpha[10]={'a','B','z','X','l','G','S','r'};
int num1[10]={1,2,3,4,5,6,8,13};
double num2[10]={2.3,4.8,3.4,19.8,6.4,5.5,10,7};
string bravo[5]={"上海","北京","沈阳","广州","武汉"};
int mian(){
>    int i=maxOfArray(num1,10);
>    cout<<"整数最大值为: "<<i<<endl;
>    double d=maxOfArray(num2,10);
>    cout<<"实数最大值为: "<<d<<endl;
>    string s=maxOfArray(bravo,5);
>    cout<<"字典排序最大为: "<<s<<endl;
>    char c=maxOfArray(alpha,10);
>    cout<<"字符最大为: "<<c<<endl;
>    return 0;
}
```

继续

```
template <typename T>void BubbleSort(T *pA, int sortedSize, int waitSortSize)
{
    bool noswap;
    int i,j;
    T temp;
    for(i=0;i<last-1;i++)
    {
        for(j=0;j<last-i-1;j++)
        {
            if(pA[j]>pA[j+1])
            {
                temp=pA[j];
                pA[j]=pA[j+1];
                pA[j+1]=temp;
                noSwap=false;
            }
        }
    if(noswap) break;
    }
};
```

唯一令人欣慰的是，
这里还知道修改……！

继续

```
#include<iostream>
#include<string>
#include"mystring.h"
using namespace std;
```

```
istream & operator >> (istream & ,MyString &ms){
    cin>>ms.str;
    return cin;
}
ostream & operator << (ostream & ,const MyString &ms){
    cout<<ms.str;
    return cout;
}
```

有同学不服气，隔壁这么写的，能正确运行。

```
ostream & operator << (ostream &cout, const MyString &ms){
    return cout<<ms.str<<'\t';}
istream & operator >> (istream &cin, MyString &ms){
    char s;
    int i=0;
    while(1){
        cin>>s;
        if(s=='\n')break;
        else{ms.str[i]=s;i++;}}
    ms.str[i]='\0';
    return cin;}
```

上上周强调过的对last的操作仍然没有！

```
char& MyString::operator[](int i)
{
    if(i>last)return str[last];
    else return str[i];
}
```

没有对last的操作！
没有越界保护！
没有更新串终止符！

实践5：类模板

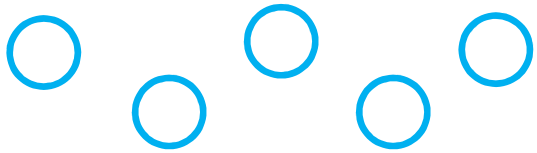
- 数据结构与算法
  - 最简单的数据结构：线性表。
    - 顺序表——使用数组顺序存储(存储方式)同一类型数据(逻辑关系)。
    - 同一种逻辑关系的数据可以有不同的存储方式。常见的线性表的存储结构：数组，链表等。
  - 算法: 操作数据的一系列方法(函数)——数据的增加，删除，修改，查找，排序等。
    - 算法的实现依赖于数据的存储方式。

实践5：类模板

- 编程思想
  - 面向过程，
    - 程序=算法+数据结构
    - 函数模板——数据通过存储地址传递给算法
  - 面向对象，
    - 程序={对象}（对象=数据+操作）
    - 类模板——数据与算法封装成类。

实践5：类模板

- Set
  - 数据元素除了同属于一个集合外，无其他关系。



- SeqList，OrderedList和后续将学的sList， dbList
  - 线性表除了是同类型数据元素的集合之外，还包含了相互之间的线性的逻辑关系，有：第一个元素，最后一个元素，前驱，后继等概念。

实践5小结

- 查漏补缺：
  - 循环结构：for (式1；式2；式3）/ while(条件) {语句块}
  - 函数：定义，声明，调用。
  - 错误：语法错误→连接错误→运行错误→逻辑错误(似是而非)
- 数组名为函数参数时的注意事项：
  - 调用时函数传入数组名
  - 数组名是只读的指针常量，系统不检查数组边界
- 数组使用注意事项(c程序员的基本素养)：
  - 使用循环访问数组时，切记检查循环的第一次和最后一次迭代，查看下标是否在允许范围内。有意识控制避免差一误差。
  - 多重循环更需要反复检查，测试阶段，每轮循环都要排查是否存在越界访问风险。

实践5小结

- 模板类相关：
  - 成员函数中缓存对象成员数据，temp的类型是T，不应是int。
  - 循环范围：0~last (在有效数据内)，不应是size.
  - 不依赖对象的函数如swap，可定义为静态成员函数。
- 函数的注意事项：
  - ☆避免上帝视角的局部域参数，主要是声明数组长度的常量。
  - dummy参数不应参与运算，若参与就直接给有意义的形参名。

实践5小结

- 练习阶段
  - 为了充分练习对数组的控制，不允许使用动态内存申请。
  - 为了体会各种排序算法的思想，除了对比验证测试结果，在自定义的排序函数中不允许调用系统自带sort()函数。
    - 学习阶段自定义的各种函数，其实都有现成库函数可调用。
  - 继续完善myString类。

实践5小结

- 程序自动化程度
  - 主要在测试阶段，综合运用已学内容，提高程序的自动化程度，人为控制越多，程序能干的事情越少。
    - 越来越多的同学使用模板来进行测试。
      - 比如线性表类增加了测试用的成员函数或函数模板
      - D2120108、16、18、19、22
      - D2120222

实践5小结

- 两种加速的排序算法(为所有努力刚过的同学加🍗🍗🍗):
  - 希尔排序比较简单，插入算法略加改造。
    - 有同学分组后采用了冒泡排序，不是希尔的思想，且增加了程序的实现难度
    - 半数同学复现了课件范例，希望自行理解，并且自己写自己调试一遍。
  - 归并排序，建议写一下引入一个新数列的算法，大部分同学能自己写出来。（👍👍致一上来就硬刚this序列但未成功的同学）
    - D2120109、15(为什么全写类内定义) 、18、23、25
    - D2120202、06、09、12、13、15、18、20、22、23、25、26
    - 还有少部分同学复现了课件范例并且测试了，👍👍可以再进一步，步子迈大一些！

函数的声明与定义时的一致性。

```
template <typename T,int size>
class seqlist
{……
    int  Find(const T & x) const;  //寻找x在表中位置（下标）
……}

//类外定义模板类的成员函数
template <typename T, int size>
int seqlist<T,size>::Find(const T & x) const
/*函数头后的const表示该函数的this指针为常量指针，即被访问对象
的数据不能被修改，如被修改，编译器会警告，防止编程时失误。*/
{……}
```

事出异常必有妖

Exp05_XXX_02_线性表类模板.cpp ×

```
istream & operator >> (istream & ifs,Student & ms)
{
    return ifs>>ms.gpa;

}
ostream & operator >> (ostream & ofs,Student & ms)
{
    return ofs<<ms.gpa;
}
```

```
friend ofstream& operator<<( ofstream& out, set<eleT,maxnum>& s
{
    int i;
    for( i=0; ;i++)
    {
        if(a.num==0) break;
        out<<a.elements[i];
        a.num--;//利用num来控制
    }
    return out;
```

愿望很美好,现实很骨感

```
// Start of your testing program
template<typename T>
void show(T &a,int size,string ss,ostream&out,int k,time_t ti) {
    cout <<endl<<endl<< ss << ":\n";
    for (int i = 0; i < size; i++) {
        cout << a[i] << ' ';
        if ((i + 1) % k == 0) cout << endl;
    }
    cout << endl;
    out <<endl<<endl<< ss << ":\n";
    for (int i = 0; i < size; i++) {
        out << a[i] << ' ';
        if ((i + 1) % k == 0) out << endl;
    }
    out << endl;
    cout << "函数运行时间: " << ti << endl;
    out << "函数运行时间: " << ti << endl;
}
```

测试时，步步为营，逐个测试好每个函数之后，再考虑附加值。神级程序员都不可能一步到位。

实践5 小结

- 若模板类set的前缀为template <typename T, int size>，则相当于 set<T, size> 为完整的类名。

```
template <typename T, int size> class set {
    T elements[size];        int num;
public:     set Intersect (set); //类内声明                    };
```

在类内，set是明确的，在编译器看到的函数声明为：

```
set<T,size> set<T,size>::Intersect ( set<T,size> )
```

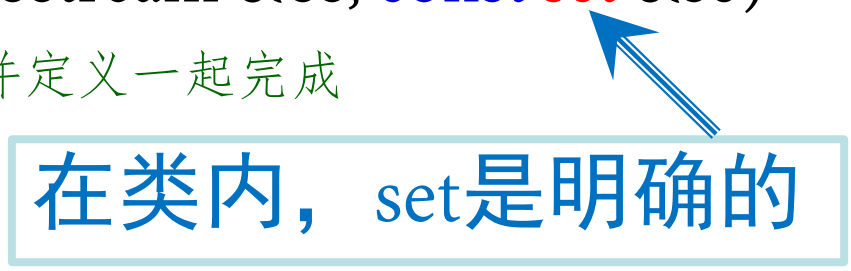
```
//类外定义
template <typename T, int size>
set<T, size> set<T, size>::Intersect(set s1)
{ set s;     …/*算法略*/      return s;    }
```

编译阶段，函数返回类型是不明确的。

实践5 小结

- 模板类重载<<时，只能在类内定义函数体。
  - 因为定义operator<<函数时，形参set<T,size>的类型是不明确的。类模板在编译阶段才将类型参数T和非类型参数size实例化。

```
template <typename T, int size> class set {
   T elements[size];
   int num;
   friend ostream& operator<<(ostream &os, const set &s0)
   { …//算法略}          //类内声明并定义一起完成
};
```



```
1  #include<iostream>
2  #include<cstring>
3  const double pi=3.14;
4  using namespace std;
5  class point  {
6  public:
7      point(int=0,int=0);
8  private:
9      double m_x;
10     double m_y;
11 };
12 class material {
13     string name;
```

Compile Log | Debug | Find Results | Close

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\temp\D1119702_2.exe
- Output Size: 1.30591678619385 MiB
```

“智能”的编译器带来的一些问题

不让程序的解释权过多依赖编译器

“智能”的编译器第2例

这是一位同学的试验程序，在不成熟阶段，故可挑刺较多，抱歉

```
set<eleT, maxnum> set<eleT,maxnum>::Union(set& s1)
{
    int i;
    set s;
    s.Copy(s1);
    for(i=0;i<this->num+s1.num;i++)
    {
        if(s.AddElem(this->elements[i]) == overflow)
        {
            s.num++;
            s.AddElem(this->elements[i]);
        }; //这里忽略了集合可能溢出
    }
    return s;
}
```

这位God设计的功能是溢出后强行写入。并且据说在DEV环境下可以运行……

实践6：模板与类参数

- 要求
  - 利用函数指针实现通用性。
  - 利用类对象作为参数，用类模板实现通用性。
  - 利用类参数传递给函数，用函数模板实现通用性。
- 编程：

1.实验17，二-2，定积分(①矩形法rectangle，②梯形法ladder，③辛普森法Simpson)的通用代码

要求:分别用函数指针, 函数模板, 类模板实现!

参考程序：割线法求根(实验5，二-3)的通用性三法。

编程要求（求根/求定积分）

- 算法见：实验五，二、3。实验17，二、2。
  - 面向过程的方式实现通用性：
    - 模仿P213,例6.13，将函数指针作为求根/定积分的函数的参数，调用时指向不同函数。
    - 模仿P211,例6.12，(模板方案)设计求根/定积分的函数模板，求根/定积分函数模板作为普通非成员函数，将待求函数定义为类，以类为模板参数。
  - 面向对象的方式实现通用性(模板方案)：
    - 模仿P210,例6.11，设计求根/定积分的类模板,割线法/辛普森法函数模板作为类模板的成员函数，待求函数通过类型参数的静态联编，引入为类模板的私有成员。

割线法求根(实验五，二、3)

设函数 f(x)定义在区间[a,b]上，f(x)连续且满足 f(a) ×f(b)<0, 求 f(x) 在[a,b]上的根。采用割线法，迭代公式为：

$$x_{i+1}= x_i+(x_{i-1}- x_i)/(f(x_i)-f(x_{i-1})) \times f(x_i)$$

其代换规律为：首先用两端点函数值的绝对值较大者的对应点作为x_{i-1}，较小者的对应点作为x_i，即如果 |f(a)|<|f(b)|，则将a赋给x_i，将b赋给x_{i-1}。用迭代公式得出x_{i+1}，$f(x_{i+1})$。

误差定义为：

$$\Delta_x =(x_{i-1}- x_i)/(f(x_i)-f(x_{i-1})) \times f(x_i)$$

当Δ_x <ε或$f(x_{i+1})=0$ 时，则结束运算。否则用$(x_i,f(x_i))$代替$(x_{i-1},f(x_{i-1}))$，$(x_{i+1},f(x_{i+1}))$代替$(x_i,f(x_i))$，继续迭代。

割线法求根(实验五，二、3)

为检验结果，必须设置下面三个函数的测试，编程输出测试结果。

计算f(x)=0在区间[2,3]内，精度为 0.00001 时的根

① f(x)= x*log10(x)-1　　//迭代次数为 4 根为 2.50618

② f(x)= 2*(x-2.288)　　//迭代次数为 1 根为 2.288

③ f(x)= 3*(x-2.5)*(x+8)　//迭代次数为 4 根为 2.5

完成了上面三个函数的测试后，可以自行构造函数，设置求根区间和精度，并完成求根测试。

近似求积分(例6.11/12/13,习题8.10)

对于函数 $f(x)$，将积分区间 [a, b] 分成 n 份，若采用等区间分割，则积分步长 $\Delta x = (b-a)/n$，每一份都是一个曲边梯形，函数 $f(x)$ 在该区间的定积分就是所有曲边梯形的面积和。

求积分的近似方法：

三种近似方法分别用直线/斜线/抛物线代替梯形的曲边，将这些曲边梯形看作近似的矩形/梯形/曲边梯形，通过计算近似矩形/梯形/曲边梯形的面积和，来近似函数在指定区间的定积分。

近似求积分(例6.11/12/13,习题8.10)

矩形法（rectangle）积分近似计算公式为：

$$\int_a^b f(x)dx \approx \Delta x(y_0 + y_1 + \cdots + y_{n-1})$$

梯形法（ladder）积分近似计算公式为：

$$\int_a^b f(x)dx \approx \frac{\Delta x}{2}(y_0 + 2(y_1 + \cdots + y_{n-1}) + y_n)$$

辛普生法（simpson）积分近似计算公式（n 为偶数）为：

$$\int_a^b f(x)dx \approx \frac{\Delta x}{3}\Big(y_0 + y_n + 4(y_1 + y_3 + \cdots + y_{n-1}) + 2(y_2 + y_4 + \cdots + y_{n-2})\Big)$$

近似求积分(例6.11/12/13,习题8.10)

编程分别用三种近似方法对下列被积函数进行定积分计算，并比较积分精度。

① **sin(x)**，　　下限为 **0.0** ，上限为**π/2**；

② **exp(x)**，　　下限为 **0.0** ，上限为 **1.0**；

③ **4.0/(1+x*x)**，　下限为 **0.0** ，上限为 **1.0**。