OOP大作业简单介绍

- 总体思路：
  - 将要序列化的对象分为基础类，复杂类，用户自定义类型和智能指针，基础类包括算数类型和string，除自定义类型和智能指针外其余为复杂类。
  - 对于基础类：
    - 将算数类型的BIT值写入文件即可。
    - 对于string，写入 其转换为cstr 的所有内容即可。
  - 对于复杂类：复杂类不只是STL里的容器，还可以是STL容器的相互嵌套版本。比如

```
vector<vector<int>>;
map < vector<map <string, int >>>
```

    - 在已经实现了基础类的序列化后，我们可以通过递归和模板特化来实现复杂类的序列化。
    - ```
template<class T> template<class Vec>void binser <T>::ser_parse(const std::vector<Vec>& myvec) {
    int type = VEC;
    file.write(reinterpret_cast<char*>(&type), sizeof(type));
    type = myvec.size();
    file.write(reinterpret_cast<char*>(&type), sizeof(type));
    for (auto iter : myvec)
    {
        ser_parse(iter);
    }
}
//list
template<class T> template<class li>void binser <T>::ser_parse(const std::list<li>& mylist) { ... }
//set
template<class T> template<class ele>void binser <T>::ser_parse(const std::set<ele>& myset) { ... }
//map
template<class T> template<class key, class key_value> void binser<T>::ser_parse(const std::map<key, key_value>&mymap) { ... }
//pair
template<class T> template<class key, class key_value> void binser<T>::ser_parse(const std::pair<key, key_value>&mypair) { ... }
//ptr
template<class T> template<class UPTR> void binser<T>::ser_parse(const std:: unique_ptr<UPTR> &unq) { ... }
//string
template<class T> void binser<T>::ser_parse(const std::string& value) { ... }
//arithmetic type
template<class T> template<class Arith>inline void binser< T>::ser_parse(const Arith& value) { ... }
```
      可以看到，对于vector,list，set，pair，map， 都可以用递归的方式来逐个序列化它们的元素。

      也不需要考虑具体该调用那一个函数，因为这是编译器该考虑的事情。

      递归到了出口就结束了。
  - 对于智能指针：其指向的内容可能是简单地，或者复杂的。
    - 我们要求用户传入外部智能指针的索引（其实接口和STL/算数类型等的序列化一样）。
    - 序列化时，只需要在开始之前向文件输出智能指针开始标志符，再调用上面的接口（将指针解引用的值当做要序列化的对象传进去），再在结束后写出智能指针结束标志符到文件即可。
    - 反序列化时，我们在内部构建一个相同类型的智能指针，将对象的值重构在内部的智能指针上。在返回之前执行以下代码即可。

```
*outterPtr=*innerPtr;
```

- 作业实现内容：
  - 要求的算数类型和STL容器；STL容器支持嵌套等复杂迭代（在测试函数里专门有测试复杂迭代的函数）。
  - unique_ptr（同样支持算数类型、string、以及STL的简单类型和复杂嵌套版本）。
```

    - base64编码的正反序列化；
- 小提示：
    - 算数类型和string以及它们的智能指针打包成一个测试函数，简单的STL类型以及它们的智能指针测试打包成一个测试函数，复杂嵌套STL类型以及它们的智能指针打包成一个测试函数，用户自定义类型打包成一个测试函数。
    - 由于使用了变参模板函数来实现用户自定义类型的序列化，导致内存上有一些意外情况，迫于时间仓促无法进一步订正o(╥﹏╥)o
        - （指如果要测试用户自定义类型的正反序列化请在程序开始时就测试，如果先运行了其它的测试函数则有概率出现段错误
        - 如果希望看到中间文件，请进入到testSer.cpp中，将所有的remove（）函数都注释掉（中间文件太多了）。默认情况下是测试完毕就删除中间文件。
        - 如果希望更改默认的用户自定义类型，请更改testSer.cpp中的mystruct结构体。并且对应修改同文件中的void testUserdefined()函数。
            - 默认的mystruct如图：

```
struct MyStruct
{
    int x;
    double y;
    vector<int> vec;
    unique_ptr<vector<double>> ptr;
    MyStruct(int in, double dou) { ... }
    bool operator==(const MyStruct& latter) {
        return(latter.x == x && latter.y == y && latter.vec == vec&&*ptr==*(latter.ptr));
    }
};
```

            - 与之配套的void testUserdefined（）函数如图：

```
void testUserdefined() {
    MyStruct test(1,1.45764657),testbin(2,2.2),testxml(3,3.3);
    vector<double> dummy;
    for (size_t i = 0; i < 12; i++)
    {
        dummy.push_back(i / 3.234);
    }
    *(test.ptr) = dummy;
    testbin.vec.clear();
    Tobin::Userialize("user", test.x, test.y, test.vec,test.ptr);
    Tobin::Udeserialize("user", 0, testbin.x, testbin.y, testbin.vec,testbin.ptr);
    testxml.vec.clear();
    Toxml::Userialize("user.xml", test.x, test.y, test.vec,test.ptr);
    Toxml::Udeserialize("user.xml", nullptr, testxml.x, testxml.y, testxml.vec,testxml.ptr);
    if ((test == testbin) && (test == testxml))
        cout << "userDefinedType passed!" << endl;
    if (!(test == testbin))
        cout << "bin for userDefiendType didn't pass!" << endl;
    if (!(test == testxml))
        cout << "xml for userDefinedType didn't pass!" << endl;
    remove("user");
    remove("user.xml");
    remove("user.base64");
}
```

            - 修改时，请注意修改mystruct中的重载的==符号，以及相关的构造函数等。
            - 对于void testUserdefined（）函数，请修改红框中的字段即可。具体地，这取决于您自定义的类型里的数据域，因为调用的是变参模板函数，此处将数据域挨个写出来即可。

```
Tobin::Userialize("user", test.x, test.y, test.vec,test.ptr);
Tobin::Udeserialize("user", 0, testbin.x, testbin.y, testbin.vec,testbin.ptr);
testxml.vec.clear();
Toxml::Userialize("user.xml", test.x, test.y, test.vec,test.ptr);
Toxml::Udeserialize("user.xml", nullptr, testxml.x, testxml.y, testxml.vec,testxml.ptr);
```

- 开发环境：vs2019，x64.