面向对象程序设计实验报告

姓名 李劼　班级 07409　学号 071202

一、 课后思考题

1. const char *func(char* const a) const 如何读？

答：名叫 func 的常成员函数，其参数为一个指向常量字符的指针，返回值为一个指向字符的常量指针。

2. C++中为什么引入 const 代替 define？

答：用 const 替换 define 指的是常量定义问题。const 比 define 有以下优点：

a) define 是预处理语句，其功能仅仅是字符串替换，而 const 常数进入编译器记号表，具有类型安全检查。

b) define 不具有封装性，一旦定义就全局可见，不能用定义一个属于某个类的常量，而 const 常数可以。

c) 用 define 定义较长的常数时，如#define pi 3.1415927，会增加目标代码长度

3. 思考如下代码的结果

```
const int i = 10;
int* const p2 = (int* const)&i;
*p2 = 111;
cout << *p2 << " " << i << endl
```

答：输出为 111 10，gcc4.4 验证。p2 是一个指向非常量整形的常量指针，而实际上 i 是个常量。如果尝试用 p2 修改 i 编译器会生成一个 i 的临时变量。

4. 思考：int (*(*f) (char *, double))[10][15]

答：一个指向 10 行 15 列二维数组的指针，数组的元素类型为返回值为 int、第一个参数为 char * 第二个参数 double 的函数指针。

二、 个人对 C++的理解

如果做横向对比，C++相对于 Java，C#，Python 等同样支持 OOP 方法的语言，效率无疑是最大的优势，指针的存在是其也拥有较高的灵活性，使其能够满足各种需要。但是 C++的标准库比较精简，使得大量第三方 C++库出现，使得程序的可维护性以及可移植性大大降低。而Java，Python，C#的程序都是通过一个虚拟机（Run Time Library）解释执行，实现平台无关，牺牲了效率，得到了可移植性。三者都拥有较为庞大的标准类库，提供了各种实用功能。而且有了 runtime 的存在，使得编译后的目标代码比较短，比较适合 web 应用。大型的本地应用还是以 C++为多，而 java，.net 的比较少（eclipse 和 netbeans 启动都至少需要 20s，大型的 java 应用还是比较恐怖的）。

C++保持了和 C 的兼容性,而系统 API 往往都是 C 接口，也就是说 C++可以直接调用系统 API，使得 C++具有了系统编程的能力。

大一寒假开始看 C++，先后看了 *primer plus*，*effective C++*，深度搜索 C++对象模型等几本关于 C++语言方面的书，还有一些 C＋＋图形化编程（MFC，GTK）的书；大概也写了几千行 C++代码。我觉得我现在困惑并不在语言本身，而在于程序结构设计方面。正如 *effective C++ Item 1* 所言：“C++是一个语言联盟”。究竟怎样运用 C++，取决于软件的设计。语言只是一个工具，设计思想才是关键。而软件设计往往都是一些经验性的方法，需要多多实践才能有感觉，自己应该多读一些源代码做积累。

三、 程序结构

平台 ：Linux x86，kernel 2.6.30-ARCH

编译器 ：gcc 4.4.0-4, gdb 6.8-3

设计的思想是：尽可能的模块化，核心代码于 UI 完全分开，给 UI 预留好接口，方便移植以及图形界面的开发。

1. 版本 1

最初的设计，将字段固定，每个字段用一个 string 存储（name，addr， phone），缺点是可扩展性差，每个字段各需要一个搜索方法、添加方法、删除方法，冗余代码多，代码重用律不高。

模块	功能	文件
Core	核心模块，记录和电话本	book.h, book.cpp
命令行界面	程序命令行界面，包括信息的显示和获取输入	cmd_interface.h cmd_interface.cpp
输入控制	检查输入、数据类型转换	input_control.h input_control.cpp

2. 版本 2

在写版本 1 的时候发现了原先设计的缺点，于是重新设计，设计成可变字段，字段在 man 函数里设定（用户接口在版本 5 中实现），用 map<string,string>存储，这样所有字段的搜索 / 添加 / 删除操作将统一（除了 index，index 是键值，类型为 int）

模块	功能	文件
Core	核心模块，记录和电话本	book.h, book.cpp
命令行界面	程序命令行界面，包括信息的显示和获取输入	cmd_interface.h cmd_interface.cpp
输入控制	检查输入、数据类型转换	input_control.h input_control.cpp

3. 版本 3

版本 3 在版本 2 的基础上增加了 Record 对象的输入输出流操作符的重载，以及文件读写。即在程序开始时从文件读出信息到内存，程序结束时把内存中的内容写回文件。

这样设计一是方便实现，另一方面，如果实时修改文件，不但插入删除非常麻烦（文件不能在中间插入或删除内容，必须全部挪动），而且一旦程序在写文件时崩溃，文件格式将发生错误，无法正确读取，全部信息丢失。

合理而安全方法应该是将修改的内容实时地按顺序地存储到到一个临时文件，程序结束时将临时文件与主文件合并。或者采用数据库来解决。

模块	功能	文件
Core	核心模块，记录和电话本	book.h, book.cpp
命令行界面	程序命令行界面，包括信息的显示和获取输入	cmd_interface.h cmd_interface.cpp
FileIO	PhoneBook 的文件读写	file_io.cpp,file_io.h

输入控制	检查输入、数据类型转换	input_control.h input_control.cpp

4. 版本 4

版本 4 在版本 3 的基础上增加了多用户支持。即每个用户拥有各自的电话本，用不同的文件存取。用户名和密码存储到一个二进制文件，并且对密码进行 md5 加密。

模块	功能	文件
Core	核心模块，记录和电话本	book.h, book.cpp
命令行界面	程序命令行界面，包括信息的显示和获取输入	cmd_interface.h cmd_interface.cpp
FileIO	PhoneBook 的文件读写	file_io.cpp,file_io.h
输入控制	检查输入、数据类型转换	input_control.h input_control.cpp
用户帐户	创建 / 验证用户	auth.cpp, auth.h
MD5	md5 加密算法	md5.cpp, md5.h

5. 版本 5

版本 5 较前面的版本改动较大，程序分为两部分：Client / Server。用户、电话本等数据存储在 server 端，client 端通过 socket 和 server 通信，获取信息。

1. socket 协议设计

#代表字段分割符，是一个无法打印的 ascii 码，而不是真的井号。程序里设置为\01。

登录认证 AUTH name# password，成功返回 true 否则 false
注册帐户 REG name# password，成功返回 true 否则 false
设置字段 FIELD num# field1# field2# field3#...field(num)
添加记录 ADD field1# field2# field3#...field(num)
编辑记录 EDIT index# filed# content
删除记录 DEL index
按索引查找 INDEX index，找到返回记录（格式见下），否则为 false
按内容查找 QUERRY field# content 找到返回记录（格式见下），否则为 false
返回记录 n record1# record2# ...# record(n)
其中 record 格式为 #field1# field2# field3#...field(num)
退出 QUIT

2．server 模块

模块	功能	文件
Core	核心模块，记录和电话本	book.h, book.cpp
用户	程序命令行界面，包括信息的显示和获取输入	cmd_interface.h cmd_interface.cpp
FileIO	PhoneBook 的文件读写	file_io.cpp,file_io.h
输入控制	数据类型转换	input_control.h input_control.cpp
用户帐户	创建 / 验证用户	auth.cpp, auth.h
MD5	md5 加密算法	md5.cpp, md5.h
请求处理	分析客户端发来的 socket，并作出相应的处理	client_process.h client_process.cpp
main	初始化 TCP socket 连接，socket/bind/listen，并给每个连接的客户端创建一个工作线程	Main.cpp

3．client 模块

模块	功能	文件
Core	只有记录，没有电话本	book.h, book.cpp
命令行界面	程序命令行界面，包括信息的显示和获取输入，将请求发往服务器，并将服务器返回的数据翻译并显示	cmd_interface.h cmd_interface.cpp
输入控制	数据类型转换	input_control.h input_control.cpp
main	初始化 TCP socket 连接到服务器, socket/connect	Main.cpp

四、 遇到的问题与收获

1. 常量对象调用非常量成员函数导致编译错误

答：比如：

```
Record::operator[](const string & field)const {
    return _fields[field];//_fields 是一个 vector<string>
}
```

编译的时候，这段代码出现了一堆匪夷所思的错误信息（其实是模板导致变量类型名过长，掩盖了真实的错误）：

```
g++ -g -c ./src/book.cpp -o ./build/book.o
./src/book.cpp: In member function 'const std::string& Record::operator[](const std::string&) const':
./src/book.cpp:56: error: passing 'const std::map<std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::less<std::basic_string<char, std::char_traits<char>, std::allocator<char> > >, std::allocator<std::pair<const std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::basic_string<char, std::char_traits<char>, std::allocator<char> > > > >' as 'this' argument of '_Tp& std::map<_Key, _Tp, _Compare, _Alloc>::operator[](const _Key&) [with _Key = std::basic_string<char, std::char_traits<char>, std::allocator<char> >, _Tp = std::basic_string<char, std::char_traits<char>, std::allocator<char> >, _Compare = std::less<std::basic_string<char, std::char_traits<char>, std::allocator<char> > >, _Alloc = std::allocator<std::pair<const std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::basic_string<char, std::char_traits<char>, std::allocator<char> > > >]' discards qualifiers
make: *** [build/book.o] Error 1
```

看了好半天才看懂，我把重载操作符定以为常成员函数，所以传进来的 this 指针就是 const 的，查了一下手册，vector 重载的[]操作符返回的是非常量引用，并且没有常量版本的重载，而 const 对象必须调用 const 成员函数，所以出现上面的问题。找到 vector 有个 find 方法有同样的功能，并且有常量版本，把上面的代码改成：

```
Record::operator[](const string & field)const {
    return (const Map &)field.find(field)->second;
}
```

就可以编译通过了。

2. 容器选择问题

容器实际上就是一种封装了算法的数据结构，不同的容器有不同的时空特性，所以选择适当的容器对程序执行效率和编写效率都有较大影响。

容器特性：

1. List 类似于链表，查找需要线性时间，添加删除消耗常数时间
2. vector 类似于数组，查找也需要线性时间，添加删除平均消耗线性时间，但可以随机存取
3. map 类似于表，内部用红黑树实现，保持有序，查找、删除、添加均消耗对数时间

选择结果：

1. phonebook 中的记录选择 List，因为 vector 在删除时消耗过多时间，而且随机存取作用并不大。
2. Record 中的字段用 map<string,string>存储，这样可以直接用字段名称快速得到字段内容。
3. PhoneBook 中的字段记录用 vector 存储，应为此部分并不经产变化，一旦初始化后就固定了。