GUI for AddressBook

1.0.0

制作者 Frank Chu

Chapter 1

README

1.1 实验三 通信录程序设计

1.1.1 一、实验目的

1. 熟悉 Qt 对话框应用程序开发的基本过程；
2. 学习数据驱动标准小部件的使用；
3. 学些文件对话框的使用；
4. 练习较复杂的交互式操作的控制流程；
5. 练习文件输入/输出流的使用。

1.1.2 二、实验内容

实现一个具有一定实用价值的通讯录程序

1.1.3 三（一）、实验系统设计

从操作逻辑中建立 view (界面对象：按钮，表格，组合框...)和 model（内部数据地址簿类与对象，相关方法，槽函数...）之间的时序逻辑关联，并设计使用槽函数将程序的各种数据结构和方法有机地串联起来。

1.1.3.1 1、通信录程序的框架结构

即给出程序设计架构图（可参考实验指导书上图，说明界面、AddressBook(类)、文件系统三者之间的关系）。

```
graph LR;
gui[对话框界面\nQTableWidget] <--槽函数--> 地址簿对象AddressBook <--QTextStream--> database[(文件系统)]
```

1.1.3.2 2、界面设计

- GroupBox 小部件
- PushButton 小部件
- LineEdit 小部件
- Label 小部件
- QTableWidget 小部件 Qt 为我们提供了两类表格小部件（QTableView 和 QTableWidget）供用户显示二维表。
- 垂直布局（Vertical Layout）在放置完所有按钮后，点选所有按钮，并点击 Vertical Layout 布局按钮进行布局。
- 组合框（Combo Box）一个 ComboBox 相当于一个 LineEdit 和一个 List 组件的结合。用于在一列数据中选择其 中某一项。一个 ComboBox 的数据可以来源于一个 QStringList 对象。

1.1.4 三（二）、主要程序说明

1.1.4.1 1、给出关键处理代码，并加以注释说明，必须包含界面上新建、更新、删除对应的功能处理代码

1. void Widget::newButtonSlot() 用于新建联系人，将新建联系人的信息添加到地址簿中，同时在界面上显示出来。
2. void Widget::alterALine() 修改某一行，更新联系人
3. void Widget::deleteARow() 删除一行数据

1.1.4.2 2、若实现了通信录排序功能，请给出相应处理代码，且在程序测试部分，给出测试结果

1. Widget::model QStandardItemModel* model 图示即为姓名降序排序

1.1.5 四、 程序测试

给出程序测试结果，并简要说明测试过程

1.1.5.1 测试结果

将文件从 Downloads 路径下读入，然后将数据显示在界面上，然后对界面上的数据进行修改，然后将修改后的数据写入文件中。

1.1.6 五、讨论及心得

1.1.6.1 1、实验过程中遇到的问题与解决方法

qt 里面 std::sort 调用别的类里面友元函数当作排序依据显示 Use of undeclared identifier

> 0x1234 At 0xFFFF搬砖艺术：
>
> lambda 早一点绑定了参数和 customSort，相当于一个闭包给 sort，不全局声明的友元函数对于他是可见的，也可以自己写个函数在类中调用 customSort 就行然后再传给 sort，还有种办法就是把友元函数的定义写在外面。
>
> 然后这里存在参数依赖查找，简称ADL，他会去Contact类里找有没有对应的函数，所以如果参数是其他的不包含Contact就找不到了。

2、目前尚未解决的问题

qt 里面 std::sort 调用别的类里面友元函数当作排序依据显示 Use of undeclared identifier

3、实验设计思路的创新等

利用 MVC 框架去对数据进行管理，将数据和界面分离，使得数据的管理更加方便。

1.1.7 六、Reference

Qt模型/视图框架（一） - 小豆君编程分享的文章 - 知乎

Qt QStandardItemModel用法（超级详细）

Chapter 2

测试列表

成员 **AddressBook::Add (CContact &contact)**

成员 **AddressBook::operator[] (int indexOfContact)**

成员 **AddressBook::Sort ()**

成员 **AddressBook::SortGroup ()**

成员 **CContact::operator< (const CContact &) const**
name = ”Frank”, number = ”1596”, Group = ”Student”

成员 **CContact::pr (const CContact &lhsContact, const CContact &rhsContact)**

成员 **match (std::string &pattern, std::string &source)**

Chapter 3

待办事项列表

成员 **Widget::Widget (QWidget *parent=nullptr)**
delete add Template data

Chapter 4

命名空间索引

4.1 命名空间列表

这里列出了所有命名空间定义,附带简要说明:

Chapter 5

继承关系索引

5.1 类继承关系

此继承关系列表按字典顺序粗略的排序:

Chapter 6

类索引

6.1 类列表

这里列出了所有类、结构、联合以及接口定义等，并附带简要说明:

Chapter 7

文件索引

7.1 文件列表

这里列出了所有文件，并附带简要说明:

Chapter 8

命名空间文档

8.1 Ui 命名空间参考

Chapter 9

类说明

9.1 AddressBook类 参考

地址簿类

`#include <AddressBook.h>`

Public 成员函数

- AddressBook ()

  Construct a new Address Book:: Address Book object 建立空地址簿
- ~AddressBook ()

  地址簿析构函数，其中必须清空联系人向量 *Book Destroy the Address Book:: Address Book object*
- void setId (int id)

  setId
- int getId () const

  getId
- void AddContact (std::string &, std::string &, std::string &)

  Add a contact in the vector<CContact> Book
- void Add (CContact &contact)

  在向量中增加一个类型为 *CContact* 联系人
- CContact operator[ ] (int indexOfContact)

  重载下标运算符
- void Sort ()

  按姓名排序 *AddressBook*
- void SortGroup ()

  按群组排序*Book*，群组名字拼音升序排序
- void List ()

  `std::cout` 输出*Book*中所有联系人。
- void ListGroup (std::string &group)

  Output all contact in
- int Delete (std::string &Name, std::string &Number, std::string &Group)

  Delete contact with Name, Number and Group, can use placeholder. 按条件删除联系人，返回删除的人数。如果没有删除任何人，返回*0*
- int Find (int startIndex, std::string &name, std::string &number, std::string &group)

  If Found, return Matched Index, else return -1. 从下标*startIndex*开始寻找符合匹配条件的联系人，如果找到，则返回下标，否则返回*-1*
- bool indexSafe (int index)

  Judge index in the range of Vector

Private 属性

- std::vector< CContact > Book

  以*CContact*类实例化类模板*vector*形成*CContact*向量作为存储结构。*Book*是*CContact*向量类的一个实例
- int id

9.1.1 详细描述

地址簿类

作者

Frank Chu

版本

v1.0.0

日期

16-Nov-2022

在文件 AddressBook.h 第 29 行定义.

9.1.2 构造及析构函数说明

9.1.2.1 AddressBook()

```
AddressBook::AddressBook ( )
```

Construct a new Address Book:: Address Book object 建立空地址簿

very good

在文件 AddressBook.cpp 第 15 行定义.

```
00015 {}
```

9.1.2.2 ~AddressBook()

```
AddressBook::~AddressBook ( )
```

地址簿析构函数，其中必须清空联系人向量 Book Destroy the Address Book:: Address Book object

在文件 AddressBook.cpp 第 17 行定义.

```
00018 {
00019     this->Book.clear();
00020 }
```

9.1.3 成员函数说明

9.1.3.1 Add()

```
void AddressBook::Add (
            CContact & contact )
```

在向量中增加一个类型为 CContact 联系人

参数

contact	输入已经建立好的 CContact 类型

测试

```
AddressBook book1;
std::string name = "Frank Chu", number = "15968126783", group = "Student";
CContact amy = CContact(name, number, group);
book1.Add(amy);
```

在文件 AddressBook.cpp 第 36 行定义.

```
00037 {
00038     this->Book.push_back(contact);
00039 }
```

9.1.3.2 AddContact()

```
void AddressBook::AddContact (
            std::string & Name,
            std::string & Number,
            std::string & Group )
```

Add a contact in the vector<CContact> Book

参数

Name	Name of Contact
Number	Number of Contact
Group	Group of Contact

在文件 AddressBook.cpp 第 22 行定义.

```
00023 {
00024     this->Book.push_back(CContact(Name, Number, Group));
00025 }
```

9.1.3.3 Delete()

```
int AddressBook::Delete (
            std::string & name,
            std::string & number,
            std::string & group )
```

Delete contact with Name, Number and Group, can use placeholder. 按条件删除联系人，返回删除的人数。如果没有删除任何人，返回0

参数

Name	Name Pattern
Number	Number Pattern
Group	Group Pattern

返回

int delete contact number

参见

AddressBook::Find()

在文件 AddressBook.cpp 第 102 行定义.

```
00103 {
00104     std::vector<int> indexOfDeletion;
00105     int deleteIndex = 0;
00106     deleteIndex = this->Find(0, name, number, group);
00107     indexOfDeletion.push_back(deleteIndex);
00108     std::cout << "Deleting " << this->Book[deleteIndex];
00109 
00110     if (deleteIndex != -1)
00111     {
00112         this->Book.erase(this->Book.begin() + deleteIndex);
00113         while (deleteIndex != -1)
00114         {
00115             // if(this->indexSafe(deleteIndex)) {
00116             deleteIndex = this->Find(0, name, number, group);
00117             if (deleteIndex != -1)
00118             {
00119                 std::cout << "Deleting " << this->Book[deleteIndex];
00120                 this->Book.erase(this->Book.begin() + deleteIndex);
00121             }
00122             // } else {
00123             //     deleteIndex = -1;
00124             // }
00125             indexOfDeletion.push_back(deleteIndex);
00126         }
00127     }
00128     // for(std::vector<CContact>::iterator itOfContacts = this->Book.begin(); itOfContacts !=
      this->Book.end(); itOfContacts++) {
00129 
00130     // }
00131     return indexOfDeletion.size() - 1;
00132 }
```

9.1.3.4 Find()

```
int AddressBook::Find (
            int startIndex,
            std::string & name,
            std::string & number,
            std::string & group )
```

If Found, return Matched Index, else return -1. 从下标startIndex开始寻找符合匹配条件的联系人，如果找到，则返回下标，否则返回-1

参数

startIndex	Start Search From index
name	name pattern
number	number pattern
group	group pattern

返回

int Matched Contact Index

在文件 AddressBook.cpp 第 41 行定义.

```
00042 {
00043     int indexOfContact = -1;
00044
00045     for (std::vector<CContact>::iterator iteratorOfContact = this->Book.begin() + startIndex;
      iteratorOfContact != this->Book.end(); iteratorOfContact++)
00046     {
00047         if ((*iteratorOfContact).PatternMatch(NamePattern, NumberPattern, GroupPattern))
00048         {
00049             indexOfContact = iteratorOfContact - Book.begin();
00050             return indexOfContact;
00051         }
00052         indexOfContact = -1;
00053     }
00054     return indexOfContact;
00055 }
```

9.1.3.5 getId()

```
int AddressBook::getId ( ) const  [inline]
```

getId

返回

在文件 AddressBook.h 第 61 行定义.

```
00061 { return this->id; }
```

9.1.3.6 indexSafe()

```
bool AddressBook::indexSafe (
             int index )
```

Judge index in the range of Vector

参数

index	My Param doc

返回

true

false

在文件 AddressBook.cpp 第 134 行定义.

```
00135 {
00136     return (index <= (int)this->Book.size()) ?  true :  false;
00137 }
```

9.1.3.7 List()

```
void AddressBook::List ( )
```

std::cout 输出Book中所有联系人。

```
// vector::begin/end
#include <iostream>
#include <vector>
int main ()
{
    std::vector<int> myvector;
    for (int i=1; i<=5; i++) myvector.push_back(i);
    std::cout << "myvector contains:";
    for (std::vector<int>::iterator it = myvector.begin() ; it != myvector.end(); ++it)
        std::cout << ' ' << *it;
    std::cout << '\n';
    return 0;
}
```

参见

public member function
std::vector::end
https://cplusplus.com/reference/vector/vector/end/

Iterator Address Book

参数

it	iterator abbreviation, *it is de-pointer of iterator(address pointer type)

在文件 AddressBook.cpp 第 218 行定义.

```
00219 {
00224     for (std::vector<CContact>::iterator it = this->Book.begin(); it != this->Book.end(); ++it)
00225     { // can also be it++
00226         std::cout << *it;
00227     }
00228 }
```

9.1.3.8 ListGroup()

```
void AddressBook::ListGroup (
            std::string & group )
```

Output all contact in

参数

group	group name

参见

- CContact::PatternMatch()
- 创建空std::string的最佳方式 http://zplutor.github.io/2016/02/18/best-way-to-create-empt

在文件 AddressBook.cpp 第 235 行定义.

```
00236 {
00237     for (std::vector<CContact>::iterator iteratorOfContact = this->Book.begin(); iteratorOfContact !=
      this->Book.end(); iteratorOfContact++)
00238     {
00239         std::string name = "", phoneNumber = "";
00240         if ((*iteratorOfContact).PatternMatch(name, phoneNumber, group))
00241         {
00242             std::cout << *iteratorOfContact;
00243         }
00244     }
00245 }
```

9.1.3.9 operator[]()

```
CContact AddressBook::operator[] (
            int indexOfContact )
```

重载下标运算符

参数

indexOfContact	index of contact

返回

CContact at specific index

测试

```
AddressBook book1;
std::string name = "Frank Chu", number = "15968126783", group = "Student";
book1.AddContact(name, number, group);
std::string rhsName = "APanda", rhsNumber = "22", rhsGroup = "Teacher";
book1.AddContact(rhsName, rhsNumber, rhsGroup);
book1.List();
std::cout << book1[0];
std::cout << book1[1];
std::cout << book1[2];
```

注解

- C++ 下标运算符 [] 重载 https://www.runoob.com/cplusplus/subscripting-operator-overload
  html
- 老师，为什么还要重载这个下标运算符啊，vector模板里不是已经实现了吗

```
int& operator[](int i)
{
    if( i >= SIZE )
    {
        cout << "索引超过最大值" <<endl;
        // 返回第一个元素
        return arr[0];
    }
    return arr[i];
}
```

在文件 AddressBook.cpp 第 86 行定义.

```
00087 {
00088     auto sizeOfBook = this->Book.size() - 1;
00089     if (indexOfContact > static_cast<int>(sizeOfBook))
00090     {
00091         std::cout << "ERROR: Larger than max index, return the first item"
00092                   << "\n";
00093         return this->Book[0];
00094     }
00095     return this->Book[indexOfContact];
00096 }
```

9.1.3.10 setId()

```
void AddressBook::setId (
             int id )  [inline]
```

setId

参数

id	

注解

In C++, the const keyword is used to specify that a function or object is intended to be a constant value and cannot be modified.

在文件 AddressBook.h 第 55 行定义.

```
00055 { this->id = id; }
```

9.1.3.11 Sort()

```
void AddressBook::Sort ( )
```

按姓名排序 AddressBook

测试

```
AddressBook book1;
std::string name = "Frank Chu", number = "15968", group = "Student";
std::string rhsName = "APanda", rhsNumber = "22", rhsGroup = "Teacher";
book1.AddContact(name, number, group);
book1.AddContact(rhsName, rhsNumber, rhsGroup);
book1.List();
book1.Sort();
book1.List();
```

参见

- C++中,结构体vector使用sort排序 https://blog.csdn.net/zhouxun623/article/details/4988755

在文件 AddressBook.cpp 第 154 行定义.

```
00155 {
00156     // std::sort(this->Book.begin(), this->Book.end(), [](const CContact& lhs, const CContact& rhs) {
      return lhs < rhs;});
00157     std::sort(this->Book.begin(), this->Book.end());
00158 }
```

9.1.3.12 SortGroup()

```
void AddressBook::SortGroup ( )
```

按群组排序Book，群组名字拼音升序排序

测试

```
AddressBook book1;
std::string name = "Frank Chu", number = "15968126783", group = "Student";
std::string rhsName = "APanda", rhsNumber = "22", rhsGroup = "Teacher";
book1.AddContact(name, number, group);
book1.AddContact(rhsName, rhsNumber, rhsGroup);
// book1.List();
// book1.Sort();
// book1.List();
CContact frank = CContact(name, number, group);
name = "Panda", number = "22", group = "Teacher";
CContact panda = CContact(name, number, group);
name = "Panda", number = "22", group = "Student";
CContact amy = CContact(name, number, group);
book1.AddContact(name, number, group);
book1.List();
book1.Sort();
book1.List();
book1.SortGroup();
book1.List();
```

在文件 AddressBook.cpp 第 185 行定义.

```
00186 {
00187 //    std::sort(this->Book.begin(), this->Book.end(), pr);
00188 //    std::sort(this->Book.begin(), this->Book.end(), [](const CContact& lhs, const CContact& rhs){
00189 //        return pr(lhs, rhs);
00190 //    });
00191     std::sort(this->Book.begin(), this->Book.end(), [](const CContact& lhs, const CContact& rhs){
00192         return CContact::prSortByGroup(lhs, rhs);
00193     });
00194 }
```

9.1.4 类成员变量说明

9.1.4.1 Book

```
std::vector<CContact> AddressBook::Book  [private]
```

以CContact类实例化类模板vector形成CContact向量作为存储结构。Book是CContact向量类的一个实例

在文件 AddressBook.h 第 35 行定义.

9.1.4.2 id

```
int AddressBook::id  [private]
```

在文件 AddressBook.h 第 36 行定义.

该类的文档由以下文件生成:

- AddressBook.h
- AddressBook.cpp

9.2 Category结构体 参考

Public 属性

- QString friends = ”Friends”
- QString colleague = ”Colleague”
- QString family = ”Family”
- QString relative = ”Relatives”
- QString student = ”Students”
- QString name = ”Name”
- QString phoneNumber = ”Phone Numebr”
- QString group = ”Group”

9.2.1 详细描述

在文件 CategoryOfAddressBook.cpp 第 3 行定义.

9.2.2 类成员变量说明

9.2.2.1 colleague

```
QString Category::colleague = "Colleague"
```

在文件 CategoryOfAddressBook.cpp 第 5 行定义.

9.2.2.2 family

```
QString Category::family = "Family"
```

在文件 CategoryOfAddressBook.cpp 第 6 行定义.

9.2.2.3 friends

```
QString Category::friends = "Friends"
```

在文件 CategoryOfAddressBook.cpp 第 4 行定义.

9.2.2.4 group

```
QString Category::group = "Group"
```

在文件 CategoryOfAddressBook.cpp 第 12 行定义.

9.2.2.5 name

```
QString Category::name = "Name"
```

在文件 CategoryOfAddressBook.cpp 第 10 行定义.

9.2.2.6 phoneNumber

```
QString Category::phoneNumber = "Phone Numebr"
```

在文件 CategoryOfAddressBook.cpp 第 11 行定义.

9.2.2.7 relative

```
QString Category::relative = "Relatives"
```

在文件 CategoryOfAddressBook.cpp 第 7 行定义.

9.2.2.8 student

```
QString Category::student = "Students"
```

在文件 CategoryOfAddressBook.cpp 第 8 行定义.

该结构体的文档由以下文件生成:

- CategoryOfAddressBook.cpp

9.3 CContact类 参考

CContact 是联系人类

```
#include <CContact.h>
```

Public 成员函数

- CContact ()
  Construct a new CContact object 默认构造函数，构造一个空联系人类
- CContact (std::string &, std::string &, std::string &)
  使用*Name,Number,Group*创建联系人对象
- CContact (const CContact &)
  拷贝构造函数
- virtual ~CContact ()
  析构函数
- void getContact (std::string &, std::string &, std::string &)
  获取对象的三个成员
- void setContact (std::string &, std::string &, std::string &)
  设定对象的三个成员
- bool operator< (const CContact &) const
  重载 < 运算符，供算法 *Sort* 使用,按姓名排序
- CContact & operator= (const CContact &)
  重载赋值 = 运算符，重载赋值运算符
- bool PatternMatch (std::string &name, std::string &number, std::string &group)
  判定本对象是否匹配搜索条件

静态 **Public** 成员函数

- static bool prSortByGroup (const CContact &lhsContact, const CContact &rhsContact)

Private 属性

- std::string Name
  姓名
- std::string Number
  电话号码
- std::string Group
  群组
- QUuid id

友元

- std::ostream & operator<< (std::ostream &, CContact)
  利用友元函数重载运算符 <<
- std::istream & operator>> (std::istream &, CContact &)
  利用友元函数重载运算符 >>
- bool pr (const CContact &lhsContact, const CContact &rhsContact)
  定义组排序函数，供算法*sort*使用，

9.3.1 详细描述

CContact 是联系人类

作者

Frank Chu

版本

v1.0.0

日期

16-Nov-2022

在文件 CContact.h 第 27 行定义.

9.3.2 构造及析构函数说明

9.3.2.1 CContact() [1/3]

```
CContact::CContact ( )
```

Construct a new CContact object 默认构造函数，构造一个空联系人类

在文件 CContact.cpp 第 14 行定义.

```
00014 {}
```

9.3.2.2 CContact() [2/3]

```
CContact::CContact (
            std::string & Name,
            std::string & Number,
            std::string & Group )
```

使用Name,Number,Group创建联系人对象

参数

Name	Contact Name
Number	Phone Number
Group	Contact Group

注解

Visual Studio Code C++ Extension July 2020 Update: Doxygen comments and Log points https://devblogs.microsoft.com/cppblog/visual-studio-code-c-extension-july-2020-update-do

在文件 CContact.cpp 第 17 行定义.

```
00017                                                   {
00018     this->Name = Name;
00019     this->Number = Number;
00020     this->Group = Group;
00021 }
```

9.3.2.3 CContact() [3/3]

```
CContact::CContact (
            const CContact & ContactInfo )
```

拷贝构造函数

参数

ContactInfo	Initialized Contact

在文件 CContact.cpp 第 23 行定义.

```
00023                                           {
00024     this->Name = ContactInfo.Name;
00025     this->Number = ContactInfo.Number;
00026     this->Group = ContactInfo.Group;
00027 }
```

9.3.2.4 ~CContact()

```
CContact::~CContact ( )  [virtual]
```

析构函数

在文件 CContact.cpp 第 29 行定义.

```
00029 { }
```

9.3.3 成员函数说明

9.3.3.1 getContact()

```
void CContact::getContact (
            std::string & Name,
            std::string & Number,
            std::string & Group )
```

获取对象的三个成员

参数

Name	return Name as reference
Number	return Number as reference
Group	return Group as reference

在文件 CContact.cpp 第 85 行定义.

```
00085                                                                    {
00086     Name = this->Name;
00087     Number = this->Number;
00088     Group = this->Group;
00089 }
```

9.3.3.2 operator<()

```
bool CContact::operator< (
            const CContact & contactToBeCompared ) const
```

重载 < 运算符，供算法 Sort 使用,按姓名排序

参数

contactToBeCompared	contact info to be compared.

返回

true thisCContact < contactToBeCompared

false thisCContact > contactToBeCompared

测试 name = "Frank", number = "1596", Group = "Student"

```
std::string name = "Frank Chu", number = "1596", group = "Student";
CContact frank = CContact(name, number, group);
name = "Panda", number = "22", group = "32";
CContact panda = CContact(name, number, group);
if(frank < panda) {
    std::cout << "<" << "\n";
} else {
    std::cout << ">" << "\n";
}
```

注解

C++ 重载运算符和重载函数

- C++ 中的运算符重载 `Box operator+(const Box&);` https://www.runoob.com/cplusplus/cpp-overloading.html
- C++使用greater报错'this' argument has type 'const xxx', but method is not marked const的解决方案 https://blog.csdn.net/HermitSun/article/details/107101944

在文件 CContact.cpp 第 50 行定义.

```
00050                                                                    {
00051     return this->Name < contactToBeCompared.Name;
00052     // return (this->Name < contactToBeCompared.Name) ?  true :  false;
00053     // if (Name < contactToBeCompared.Name) {
00054     //     return true;
00055     // } else {
00056     //     return false;
00057     // }
00058 }
```

9.3.3.3 operator=()

```
CContact & CContact::operator= (
            const CContact & oldContact )
```

重载赋值 = 运算符，重载赋值运算符

参数

oldContact	local variable, newContact = oldContact, set new is equal to old.

注解

C++ 赋值运算符 = 重载 https://www.runoob.com/cplusplus/assignment-operators-overloading.html

在文件 CContact.cpp 第 61 行定义.

```
00061                                                   {
00062     this->Name = oldContact.Name;
00063     this->Number = oldContact.Number;
00064     this->Group = oldContact.Group;
00065     return *this;
00066 }
```

9.3.3.4 PatternMatch()

```
bool CContact::PatternMatch (
            std::string & NamePattern,
            std::string & NumberPattern,
            std::string & GroupPattern )
```

判定本对象是否匹配搜索条件

参数

name	name pattern
number	number pattern
group	group pattern

返回

true 符合
false 不符合

参见

C++ 函数默认参数 https://www.w3cschool.cn/cpp/cpp-function-default-parameters.html

在文件 CContact.cpp 第 101 行定义.

```
00101                                                                                {
00102     return (
00103         match(NamePattern, this->Name) && match(NumberPattern, this->Number) && match(GroupPattern,
      this->Group)
00104     ) ?  true :  false;
00105 }
```

9.3.3.5 prSortByGroup()

```
static bool CContact::prSortByGroup (
              const CContact & lhsContact,
              const CContact & rhsContact )  [inline], [static]
```

在文件 CContact.h 第 108 行定义.

```
00108                                                                                {
00109         return lhsContact.Group < rhsContact.Group;
00110     }
```

9.3.3.6 setContact()

```
void CContact::setContact (
              std::string & Name,
              std::string & Number,
              std::string & Group )
```

设定对象的三个成员

参数

Name	Contact Name
Number	Phone Number
Group	Contact Group

在文件 CContact.cpp 第 91 行定义.

```
00091                                                                                {
00092     this->Name = Name;
00093     this->Number = Number;
00094     this->Group = Group;
00095 }
```

9.3.4 友元及相关函数文档

9.3.4.1 operator<<

```
std::ostream & operator<< (
              std::ostream & os,
              CContact contactInfo )  [friend]
```

利用友元函数重载运算符 <<

参数

std::ostream	file stream
CContact	output contact class

返回

ostream

参见

【懒猫老师-最简版C++-(18)类的友元】 https://www.bilibili.com/video/BV127411↩Q7eu/

注解

Overloading the << Operator for Your Own Classes https://learn.microsoft.com/en-us/cpp/standard-li

在文件 CContact.cpp 第 70 行定义.

```
00070                                                 {
00071     // Overloading the >> Operator for Your Own Classes
00072     os << contactInfo.Name << ", " << contactInfo.Number << ", " << contactInfo.Group << "\n";
00073     return os;
00074 }
```

9.3.4.2 operator>>

```
std::istream & operator>> (
            std::istream & is,
            CContact & contactToBeRevised )  [friend]
```

利用友元函数重载运算符 >>

参数

std::istream&	is
CContact	

返回

istream

注解

Overloading the >> Operator for Your Own Classes https://learn.microsoft.com/en-us/cpp/standard-li

在文件 CContact.cpp 第 77 行定义.

```
00077                                                 {
```

```
00078     std::string name, number, group;
00079     is >> name >> number >> group;
00080     contactToBeRevised.setContact(name, number, group);
00081     // std::cout << name << number << group;
00082     return is;
00083 }
```

9.3.4.3 pr

```
bool pr (
              const CContact & lhsContact,
              const CContact & rhsContact )  [friend]
```

定义组排序函数，供算法sort使用,

参数

lhsContact	left hand side of Contact to be compared
rhsContact	right hand side of Contact to be compared

返回

true lhsContact.Group < rhsContact.Group;

false lhsContact.Group > rhsContact.Group;

测试

```
std::string name = "Frank Chu", number = "1596", group = "Student";
std::string rhsName = "Panda", rhsNumber = "22", rhsGroup = "Teacher";
if(pr(CContact(name, number, group), CContact(rhsName, rhsNumber, rhsGroup))) {
    std::cout << "< change postion" << "\n";
} else {
    std::cout << "> do not change" << "\n";
}
```

在文件 CContact.cpp 第 160 行定义.

```
00160                                                                    {
00161     return lhsContact.Group < rhsContact.Group;
00162 }
```

9.3.5 类成员变量说明

9.3.5.1 Group

```
std::string CContact::Group  [private]
```

群组

在文件 CContact.h 第 32 行定义.

9.3.5.2 id

```
QUuid CContact::id  [private]
```

在文件 CContact.h 第 33 行定义.

9.3.5.3 Name

```
std::string CContact::Name  [private]
```

姓名

在文件 CContact.h 第 30 行定义.

9.3.5.4 Number

```
std::string CContact::Number  [private]
```

电话号码

在文件 CContact.h 第 31 行定义.

该类的文档由以下文件生成:

- CContact.h
- CContact.cpp

9.4 Widget类 参考

```
#include <widget.h>
```

Public 成员函数

- Widget (QWidget *parent=nullptr)

  Widget::Widget
- ~Widget ()
- void Bind ()

  Bind bind signal and slots
- void fillComboBoxAndTableWidget ()

  fillComboBoxAndTableWidget fill with 1) 设定列标题及样式;
- void keyPressEvent (QKeyEvent *key)

  keyPressEvent shortcut for load(ctrl+o) save(ctrl+s)
- void addTemplateData ()
- void clearLineEdit ()

  clearLineEdit
- void initModelFromStringList (QStringList &fileContentStringList)

  initModelFromStringList 调用自定义函数 *initModelFromStringList()*，用 *fileContentStringList* 的内容初始化数据模型
- void customizedTableStyle ()

  customizedTableStyle 设定自定义表的样式

Public 属性

- AddressBook addressBook

Private 槽

- void ExitWidget ()
- void LoadFile ()

  LoadFile 8、 文件读取操作流程框架 "打开文件"按钮的槽函数代码
- void saveButtonSlot ()

  saveButtonSlot 9、 文件保存流程框架
- void newButtonSlot ()

  newButtonSlot 2) 加入一行数据;
- void clickedTableView ()

  clickedTableView 3) 选中一行数据（槽函数）;
- void deleteARow ()

  deleteARow 4) 删除一行数据;
- void alterALine ()

  alterALine 5) 修改某一行;

Private 属性

- Category categories

  categories category for comboBox
- Ui::Widget * ui
- QStandardItemModel * model

  model

9.4.1 详细描述

在文件 widget.h 第 20 行定义.

9.4.2 构造及析构函数说明

9.4.2.1 Widget()

```
Widget::Widget (
            QWidget * parent = nullptr )
```

Widget::Widget

参数

parent	

注解

In this code, Widget is a class that is derived from the QWidget class. The Widget class has a constructor that takes a QWidget pointer as an argument and assigns it to the parent of the new Widget object. The constructor also initializes a new Ui::Widget object and assigns it to the ui member variable of the Widget object.

待办事项 delete add Template data

在文件 widget.cpp 第 14 行定义.

```
00014                               :  QWidget(parent) ,
00015     ui(new Ui::Widget)
00016 {
00017     this->ui->setupUi(this);
00018     this->Bind();
00019     this->fillComboBoxAndTableWidget();
00020 //    this->addTemplateData();
00021 }
```

9.4.2.2 ~Widget()

```
Widget::~Widget ( )
```

在文件 widget.cpp 第 23 行定义.

```
00024 {
00025     delete this->ui;
00026 }
```

9.4.3 成员函数说明

9.4.3.1 addTemplateData()

```
void Widget::addTemplateData ( )
```

在文件 widget.cpp 第 279 行定义.

```
00280 {
00281     QStandardItem* name = new QStandardItem(QString("apple"));
00282     QStandardItem* phone = new QStandardItem(QString("400-666-8800"));
00283     QStandardItem* group = new QStandardItem(QString("Colleague"));
00284
00285     QList<QStandardItem*> itemList;
00286     itemList << name << phone << group;
00287     model->appendRow(itemList);
00288
00289     name = new QStandardItem(QString("google"));
00290     phone = new QStandardItem(QString("901-3283-2233"));
00291     group = new QStandardItem(QString("Colleague"));
00292     itemList.clear();
00293     itemList << name << phone << group;
00294     model->appendRow(itemList);
00295 }
```

9.4.3.2 alterALine

```
void Widget::alterALine ( )  [private], [slot]
```

alterALine 5) 修改某一行;

在文件 widget.cpp 第 441 行定义.

```
00442 {
00443     // 将数据先保存到地址簿当下位置
00444     // 获取当前联系人行
00445     this->newButtonSlot();
00446     this->deleteARow();
00447 }
```

9.4.3.3 Bind()

```
void Widget::Bind ( )
```

Bind bind signal and slots

在文件 widget.cpp 第 28 行定义.

```
00028                 {
00029     QObject::connect(ui->nameLineEdit, SIGNAL(returnPressed()), this, SLOT(ExitWidget()));
00030     QObject::connect(ui->exitButton, SIGNAL(clicked(bool)), this, SLOT(ExitWidget()));
00031 //    QObject::connect(ui->exitButton, &QPushButton::clicked, this, &Widget::ExitWidget);
00032 //    QObject::connect(ui->exitButton, SLGNAL(clicked()), this, SLOT(ExitWidget()));
00033 //    connect(ui->loadButton, &QPushButton::clicked, [this](){
00034 //        QMessageBox::information(this, "Info", "Click to explore");
00035 //    });
00036
00037 //    QObject::connect(ui->loadButton, SIGNAL(clicked()), this, SLOT(LoadFile()));
00038
00039     QObject::connect(this->ui->newButton, &QPushButton::clicked, this, &Widget::newButtonSlot);
00040     QObject::connect(this->ui->loadButton, &QPushButton::clicked, this, &Widget::LoadFile);
00041     QObject::connect(this->ui->saveButton, &QPushButton::clicked, this, &Widget::saveButtonSlot);
00042     QObject::connect(this->ui->tableView, &QTableView::clicked, this, &Widget::clickedTableView);
00043     QObject::connect(this->ui->deleteButton, &QPushButton::clicked, this, &Widget::deleteARow);
00044     QObject::connect(this->ui->updateButton, &QPushButton::clicked, this, &Widget::alterALine);
00045
00046 }
```

9.4.3.4 clearLineEdit()

```
void Widget::clearLineEdit ( )
```

clearLineEdit

在文件 widget.cpp 第 297 行定义.

```
00298 {
00299     // clear Text
00300     this->ui->nameLineEdit->setText("");
00301     this->ui->phoneLineEdit->setText("");
00302 }
```

9.4.3.5 clickedTableView

```
void Widget::clickedTableView ( )  [private], [slot]
```

clickedTableView 3) 选中一行数据（槽函数）；

在文件 widget.cpp 第 408 行定义.

```
00409 {
00410     QAbstractItemModel* modelSelected = this->ui->tableView->model();
00411     int row = this->ui->tableView->currentIndex().row();
00412 //    qDebug() << row;
00413
00414     //根据行号在地址簿中读取并显示相应的联系人至编辑界面
00415     this->ui->nameLineEdit->setText(modelSelected->index(row, 0).data().toString());
00416     this->ui->phoneLineEdit->setText(modelSelected->index(row, 1).data().toString());
00417     this->ui->groupComboBox->setCurrentText(modelSelected->index(row, 2).data().toString());
00418
00419     //选中一行后启用按钮
00420     this->ui->deleteButton->setEnabled(true);
00421     this->ui->updateButton->setEnabled(true);
00422 }
```

9.4.3.6 customizedTableStyle()

```
void Widget::customizedTableStyle ( )
```

customizedTableStyle 设定自定义表的样式

在文件 widget.cpp 第 382 行定义.

```
00383 {
00384     // Style sheet
00385     //设定列延展特性
00386     this->ui->tableWidget->horizontalHeader()->setStretchLastSection(true);
00387     this->ui->tableView->horizontalHeader()->setStretchLastSection(true);
00388
00389     // QTableView column width
00390     // https://stackoverflow.com/questions/26681578/qtableview-column-width
00391     this->ui->tableView->setColumnWidth(1, 200);
00392
00393     //设定标题样式
00394     this->ui->tableWidget->horizontalHeader()->setStyleSheet("border:none; border-bottom:1px solid
      grey");
00395     this->ui->tableView->horizontalHeader()->setStyleSheet("border:none; border-bottom:1px solid
      grey");
00396
00397     //设定表格行选定行为
00398     this->ui->tableWidget->setSelectionMode(QAbstractItemView::SingleSelection);
00399     this->ui->tableWidget->setSelectionBehavior(QAbstractItemView::SelectRows);
00400
00401     this->ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
00402
00403     //显示网格线
00404     this->ui->tableWidget->showGrid();
00405     this->ui->tableView->showGrid();
00406 }
```

9.4.3.7 deleteARow

```
void Widget::deleteARow ( )  [private], [slot]
```

deleteARow 4) 删除一行数据；

参见

Qt Delete selected row in QTableView https://stackoverflow.com/questions/19012450/qt-delete-sel

在文件 widget.cpp 第 427 行定义.

```
00428 {
00429     // 获取当前联系人行
00430     auto currentRow = this->ui->tableView->currentIndex().row();
00431
00432     // 将联系人从数据模型中删除
00433     this->model->removeRow(currentRow);
00434
00435     // 调整相关按钮 LineEdit 状态
00436     this->clearLineEdit();
00437     this->ui->deleteButton->setEnabled(false);
00438     this->ui->updateButton->setEnabled(false);
00439 }
```

9.4.3.8 ExitWidget

```
void Widget::ExitWidget ( )  [private], [slot]
```

参见

【QT快速入门 | 最简单最简洁的QT入门教程 | 嵌入式UI-哔哩哔哩】 https://b23.tv/ADss8yy

在文件 widget.cpp 第 260 行定义.

```
00261 {
00262 //    QString program = "/System/Applications/Utilities/Terminal.app";
00263 //    program = "/Applications/" + this->ui->nameLineEdit->text() + ".app";
00264 //    this->ui->nameLineEdit->setText(program);
00265 //    QProcess *myProcess = new QProcess(this);
00266 //    myProcess->start(program);
00267     this->close();
00268 }
```

9.4.3.9 fillComboBoxAndTableWidget()

```
void Widget::fillComboBoxAndTableWidget ( )
```

fillComboBoxAndTableWidget fill with 1) 设定列标题及样式;

注解

一个 ComboBox 相当于一个 LineEdit 和一个 List 组件的结合。用于在一列数据中选择其中某一项。一个 ComboBox 的数据可以来源于一个 QStringList 对象。

1. 填充列表数据;
2. 设定当前选项数据;
3. 获取当前选项数据;

QTableWidget 小部件的使用 Qt 为我们提供了两类表格小部件（QTableView 和 QTableWidget）供用户显示二维表。 QTableView 基于 View/Model/Delegate 的灵活架构，可以非常灵活地以多种方式显示后台数据（model）； 而 QTableWidget 也是基于 View/Model 架构，但它提供了默认的数据结构，用户不需要涉及过多的架构细节就可以迅速地用它来显示二维数据表格。 QTableWidget 使用行号（row）和列号（column）去定位一个单元格（cell）并显示数据。

参见

C++ Qt 47 - Intro to model view programming https://www.youtube.com/watch?v=uDC9L4↩T59bM

在文件 widget.cpp 第 64 行定义.

```
00065 {
00066
00067 //    this->ui->groupComboBox->setModel();
00068 //    this->ui->groupComboBox->setModel();
00069     this->addressBook.setId(0);
00070
00071     // ComboBox
00072     QStringList groupItems;
00073     groupItems.append(categories.friends);
00074     groupItems.append(categories.colleague);
00075     groupItems.append(categories.family);
00076     groupItems.append(categories.relative);
00077     groupItems.append(categories.student);
00078     this->ui->groupComboBox->addItems(groupItems);
00079
00080     this->ui->groupComboBox->setCurrentText(categories.student);
00081
00082 //    qDebug() << this->ui->groupComboBox->currentText();
00083
00084     // QTableWidget 小部件的使用
00085     QStringList header;
00086     header << categories.name << categories.phoneNumber << categories.group;
00087     this->ui->tableWidget->setColumnCount(3); //设定列总数为  3
00088     this->ui->tableWidget->setHorizontalHeaderLabels(header); //设定列标题
00089
00090     // Table View(Model Based) Title
00091     this->model = new QStandardItemModel;
00092     this->model->setHorizontalHeaderLabels(header);
00093
00094     // Table View(Model Based) model for view
00095     this->ui->tableView->setModel(model);
00096     this->ui->tableWidget->hide();
00097 //    this->ui->tableView->hide();
00098
00099     this->customizedTableStyle();
00100 }
```

9.4.3.10 initModelFromStringList()

```
void Widget::initModelFromStringList (
            QStringList & fileContentStringList )
```

initModelFromStringList 调用自定义函数 initModelFromStringList()，用 fileContentStringList 的内容初始化数据模型

参数

fileContentStringList	

注解

传递来的参数 fileContentStringList 是文本文件所有行构成的 StringList，文件的每一行是 aFileContent 的一行字符串，第 1 行是表头文字，数据从第 2 行开始。程序首先获取字符串列表的行数，然后设置数据模型的行数，因为数据模型的列数在初始化时已经设置了。然后获取字符串列表的第 1 行，即表头文字，用 QString::split() 函数分割成一个 QStringList，设置为数据模型的表头标题。QString::split() 函数根据某个特定的符号将字符串进行分割。例如，header 是数据列的标题，每个标题之间通过一个或多个 TAB 键分隔，其内容是：

```
Name        Phone Number        Group
```

那么通过上面的 split() 函数操作，得到一个字符串列表 headerList，其内容是：

```
Name
Phone Number
Group
```

也就是分解为一个 3 行的 StringList。然后使用此字符串列表作为数据模型，设置表头标题的函数 setHorizontalHeaderLabels() 的参数，就可以为数据模型设置表头了。 同样，在逐行获取字符串后，也采用 split() 函数进行分解，为每个数据创建一个 QStandardItem 类型的项数据 aItem，并赋给数据模型作为某行某列的项数据。 QStandardItemModel 以二维表格的形式保存项数据，每个项数据对应着 QTableView 的一个单元格。项数据不仅可以存储显示的文字，还可以存储其他角色的数据。

multiTab

注解

Description: Don't create temporary QRegularExpression objects. a Use a static QRegularExpression object instead [clazy-use-static-qregularexpression] Location: /Users/yongfrank/Developer/Cpp/lab/↩ Cpp-labo3-week14/AddressBookGU/widget.cpp:360:43 Documentation: https://github.com/↩ KDE/clazy/blob/1.11/docs/checks/READ ME-use-statio-qregularexpression.md

在文件 widget.cpp 第 322 行定义.

```
00323 {
00324     // 将内存中的 model 数据模型清除
00325     this->model->clear();
00326
00327     // 从一个StringList 获取数据，初始化数据Model
00328     // 文本行数，第1行是标题
00329     int rowCount = fileContentStringList.count();
00330
00331     // 实际数据行数
00332     this->model->setRowCount(rowCount - 1);
00333
00334     // 设置表头
00335     // 第1行是表头
00336     QString header = fileContentStringList.at(0);
00337
00349     static auto multiTab = QRegularExpression("\\t+");
00350
00351     // 一个或多个空格、TAB("\s+") 多个 TAB 分隔("\t+")等分隔符隔开的字符串， 分解为一个StringList
00352     QStringList headerList = header.split(multiTab);
00353     headerList.removeLast();
00354     //设置表头文字
00355     this->model->setHorizontalHeaderLabels(headerList);
00356
00357     //设置表格数据
00358 //    QString lineText;
00359 //    QStringList lineTempList;
00360     QStandardItem *itemStandard;
00361     for(int i = 0; i < rowCount; i++) {
00362
00363         //获取数据区的一行
00364         QString lineText = fileContentStringList.at(i);
00365
00366         //一个或多个空格、TAB等分隔符隔开的字符串， 分解为一个StringList
00367         QStringList lineTempList = lineText.split(multiTab);
00368
00369         //lineTempList 的行数等于 headerList.count, 固定的
00370         for(int j = 0; j < headerList.count(); j++) {
00371
00372             //创建item
00373             itemStandard = new QStandardItem(lineTempList.at(j));
00374
00375             //为模型的某个行列位置设置Item
00376             this->model->setItem(i - 1, j, itemStandard);
00377         }
00378     }
00379     this->customizedTableStyle();
00380 }
```

9.4.3.11 keyPressEvent()

```
void Widget::keyPressEvent (
            QKeyEvent * key )
```

keyPressEvent shortcut for load(ctrl+o) save(ctrl+s)

参数

key	

在文件 widget.cpp 第 270 行定义.

```
00270                                       {
00271     if(key->modifiers() == Qt::ControlModifier && key->key() == Qt::Key_S) {
00272         this->saveButtonSlot();
00273     }
00274     if(key->modifiers() == Qt::ControlModifier && key->key() == Qt::Key_O) {
00275         this->LoadFile();
00276     }
00277 }
```

9.4.3.12 LoadFile

```
void Widget::LoadFile ( )  [private], [slot]
```

LoadFile 8、 文件读取操作流程框架 “打开文件”按钮的槽函数代码

注解

”/home”, ”*.txt, *.cpp” 这段代码让用户选择所需要打开的数据文本文件，然后用只读和文本格式打开文件，逐行读取其内容，将每行字符串显示到界面上的 TextEdit 里，并且添加到一个临时的 QStringList 类型的变量 FileContent 里。

参见

Qt QStandardItemModel用法（超级详细） http://c.biancheng.net/view/1869.html

在文件 widget.cpp 第 134 行定义.

```
00134                       {
00135
00136     // 获取应用程序的路径
00137     // 调用打开文件对话框打开一个文件
00138     QString fileName = QFileDialog::getOpenFileName(this, "Choose a file",
      QApplication::applicationDirPath(),
00139                                                     "Text File(*.txt);;All File(*.*)");
00140 //    QString fileName = QFileDialog::getOpenFileName(this, "Choose a File",
00141 //                                                    QCoreApplication::applicationFilePath());
00142 //    QFileDialog::getOpenFileName(this, "Choose a File", QCoreApplication::applicationDirPath());
00143
00144     // 如果未选择文件，退出
00145     if (fileName.isEmpty()) {
00146         QMessageBox::warning(this, "Warning", "Choose a File, please");
00147         return;
00148     }
00149 //    qDebug() << fileName;
00150
00151     // 文件内容字符串列表
00152     QStringList fileContentStringList;
00153
00154     // Create a QFile Object
00155     // https://doc.qt.io/qt-6/qfile.html
00156     // 以文件方式读出
00157     QFile file(fileName);
00158
00159     // 以只读文本方式打开文件
00160     file.open(QIODevice::ReadOnly);
00161
00162     // 用文本流读取文件
00163     QTextStream fileStream(&file);
00164
00165     // 清空
00166     this->ui->textEdit->clear();
00167
```

```
00168     while(!fileStream.atEnd()) {
00169         // 读取文件的一行
00170         QString stringToBeDisplayed = fileStream.readLine();
00171 
00172         // 添加到文本框显示
00173         this->ui->textEdit->append(stringToBeDisplayed);
00174 
00175         // 添加到 fileContentStringList
00176         fileContentStringList.append(stringToBeDisplayed);
00177     }
00178 
00179     // 关闭文件
00180     //    QByteArray byteArray = file.readAll();
00181     //    ui->textEdit->setText(QString(byteArray));
00182     file.close();
00183 
00184     // 窗口标题显示
00185     this->setWindowTitle(fileName);
00186 
00187     // 从StringList的内容初始化数据模型
00188     this->initModelFromStringList(fileContentStringList);
00189 }
```

9.4.3.13 newButtonSlot

```
void Widget::newButtonSlot ( )  [private], [slot]
```

newButtonSlot 2) 加入一行数据;

在文件 widget.cpp 第 102 行定义.

```
00102                           {
00103 //    this->ui->textEdit->clear();
00104 //    this->setWindowTitle("New Address Book");
00105 //    QUuid id = QUuid();
00106 
00107     // 获得行计数
00108 //    int lastRow = this->model->rowCount();
00109 //    qDebug() << lastRow;
00110 
00111     std::string name = this->ui->nameLineEdit->text().toStdString();
00112     std::string phone = this->ui->phoneLineEdit->text().toStdString();
00113     std::string group = this->ui->groupComboBox->currentText().toStdString();
00114 //    qDebug() << id << QString::fromStdString(name) << this->ui->groupComboBox->currentText();
00115 
00116     this->addressBook.AddContact(name, phone, group);
00117     QStandardItem* nameItem = new QStandardItem(QString::fromStdString(name));
00118     QStandardItem* phoneItem = new QStandardItem(QString::fromStdString(phone));
00119     QStandardItem* groupItem = new QStandardItem(QString::fromStdString(group));
00120 
00121     QList<QStandardItem*> itemList;
00122     itemList << nameItem << phoneItem << groupItem;
00123 
00124     // 在表格后添加一行
00125     model->appendRow(itemList);
00126 
00127     this->clearLineEdit();
00128 }
```

9.4.3.14 saveButtonSlot

```
void Widget::saveButtonSlot ( )  [private], [slot]
```

saveButtonSlot 9、 文件保存流程框架

参见

https://doc.qt.io/qt-6/qbytearray.html Qt QStandardItemModel用法（超级详细） http://c.biancheng.net/view/1869.html

在文件 widget.cpp 第 195 行定义.

```
00195                           {
00196
00197     // 获取应用程序的路径
00198 //    QString curPath = QCoreApplication::applicationDirPath();
00199 //    qDebug() << curPath;
00200
00201     // Save New Dialog
00202     // 调用打开文件对话框选择一个文件
00203     QString fileName = QFileDialog::getSaveFileName(this, "Chose a file",
      QCoreApplication::applicationFilePath(),
00204                                                     "Text File(*.txt);;All File(*.*)");
00205
00206
00207     // 未选择文件，退出
00208     if (fileName.isEmpty()) {
00209         QMessageBox::warning(this, "Warning", "Choose a file");
00210         return;
00211     }
00212
00213     QFile file(fileName);
00214     file.open(QIODevice::WriteOnly);
00215
00216 //    QByteArray ba;
00217 //    ba.append(this->ui->textEdit->toPlainText().toUtf8());
00218 //    file.write(ba);
00219 //    file.close();
00220
00221     QString stringToBeStored;
00222
00223     // 用文本流读取文件
00224     QTextStream fileStream(&file);
00225     QStandardItem* item;
00226
00227     // 获取表头文字
00228     for(int i = 0; i < this->model->columnCount(); i++) {
00229
00230         // 获取表头的项数据
00231         item = this->model->horizontalHeaderItem(i);
00232
00233         // 以 TAB 间隔开
00234         stringToBeStored += item->text() + "\t\t";
00235     }
00236
00237     // 文件里需要加入换行符 \n
00238     fileStream << stringToBeStored << "\n";
00239
00240     this->ui->textEdit->append(stringToBeStored);
00241
00242     // 获取数据区文字
00243     for (int i = 0; i < this->model->rowCount(); i++) {
00244         stringToBeStored = "";
00245         for (int j = 0; j < this->model->columnCount(); j++) {
00246             item = this->model->item(i, j);
00247             stringToBeStored += item->text() + "\t\t";
00248         }
00249
00250         this->ui->textEdit->append(stringToBeStored);
00251         fileStream << stringToBeStored << "\n";
00252     }
00253
00254     file.close();
00255 }
```

9.4.4 类成员变量说明

9.4.4.1 addressBook

AddressBook Widget::addressBook

在文件 widget.h 第 28 行定义.

9.4.4.2 categories

```
Category Widget::categories  [private]
```

categories category for comboBox

在文件 widget.h 第 101 行定义.

9.4.4.3 model

```
QStandardItemModel* Widget::model  [private]
```

model

参见

Qt模型/视图框架（一） - 小豆君编程分享的文章 - 知乎 https://zhuanlan.zhihu.com/p/47402006 Address Book Example https://doc.qt.io/qt-6/qtwidgets-itemviews-addressbook.html

在文件 widget.h 第 110 行定义.

9.4.4.4 ui

```
Ui::Widget* Widget::ui  [private]
```

在文件 widget.h 第 103 行定义.

该类的文档由以下文件生成:

- widget.h
- widget.cpp

Chapter 10

文件说明

10.1 AddressBook.cpp 文件参考

```
#include "AddressBook.h"
```

10.2 AddressBook.cpp

浏览该文件的文档.

```
00001 /*
00002 * @Author:  Frank Chu
00003 * @Date:  2022-11-17 23:42:03
00004 * @LastEditors:  Frank Chu
00005 * @LastEditTime:  2022-11-22 14:05:23
00006 * @FilePath:  /Cpp/lab/Cpp-lab01-week11/source/AddressBook.cpp
00007 * @Description:
00008 *
00009 * Copyright (c) 2022 by Frank Chu, All Rights Reserved.
00010 */
00011
00012 #include "AddressBook.h"
00013
00015 AddressBook::AddressBook() {}
00016
00017 AddressBook::~AddressBook()
00018 {
00019     this->Book.clear();
00020 }
00021
00022 void AddressBook::AddContact(std::string &Name, std::string &Number, std::string &Group)
00023 {
00024     this->Book.push_back(CContact(Name, Number, Group));
00025 }
00026
00036 void AddressBook::Add(CContact &contact)
00037 {
00038     this->Book.push_back(contact);
00039 }
00040
00041 int AddressBook::Find(int startIndex, std::string &NamePattern, std::string &NumberPattern,
      std::string &GroupPattern)
00042 {
00043     int indexOfContact = -1;
00044
00045     for (std::vector<CContact>::iterator iteratorOfContact = this->Book.begin() + startIndex;
      iteratorOfContact != this->Book.end(); iteratorOfContact++)
00046     {
00047         if ((*iteratorOfContact).PatternMatch(NamePattern, NumberPattern, GroupPattern))
00048         {
00049             indexOfContact = iteratorOfContact - Book.begin();
00050             return indexOfContact;
00051         }
```

```
00052         indexOfContact = -1;
00053     }
00054     return indexOfContact;
00055 }
00056
00086 CContact AddressBook::operator[](int indexOfContact)
00087 {
00088     auto sizeOfBook = this->Book.size() - 1;
00089     if (indexOfContact > static_cast<int>(sizeOfBook))
00090     {
00091         std::cout << "ERROR: Larger than max index, return the first item"
00092                   << "\n";
00093         return this->Book[0];
00094     }
00095     return this->Book[indexOfContact];
00096 }
00097
00102 int AddressBook::Delete(std::string &name, std::string &number, std::string &group)
00103 {
00104     std::vector<int> indexOfDeletion;
00105     int deleteIndex = 0;
00106     deleteIndex = this->Find(0, name, number, group);
00107     indexOfDeletion.push_back(deleteIndex);
00108     std::cout << "Deleting " << this->Book[deleteIndex];
00109
00110     if (deleteIndex != -1)
00111     {
00112         this->Book.erase(this->Book.begin() + deleteIndex);
00113         while (deleteIndex != -1)
00114         {
00115             // if(this->indexSafe(deleteIndex)) {
00116             deleteIndex = this->Find(0, name, number, group);
00117             if (deleteIndex != -1)
00118             {
00119                 std::cout << "Deleting " << this->Book[deleteIndex];
00120                 this->Book.erase(this->Book.begin() + deleteIndex);
00121             }
00122             // } else {
00123             //     deleteIndex = -1;
00124             // }
00125             indexOfDeletion.push_back(deleteIndex);
00126         }
00127     }
00128     // for(std::vector<CContact>::iterator itOfContacts = this->Book.begin(); itOfContacts !=
    this->Book.end(); itOfContacts++) {
00129
00130     // }
00131     return indexOfDeletion.size() - 1;
00132 }
00133
00134 bool AddressBook::indexSafe(int index)
00135 {
00136     return (index <= (int)this->Book.size()) ?  true :  false;
00137 }
00138
00154 void AddressBook::Sort()
00155 {
00156     // std::sort(this->Book.begin(), this->Book.end(), [](const CContact& lhs, const CContact& rhs) {
    return lhs < rhs;});
00157     std::sort(this->Book.begin(), this->Book.end());
00158 }
00159
00185 void AddressBook::SortGroup()
00186 {
00187 //    std::sort(this->Book.begin(), this->Book.end(), pr);
00188 //    std::sort(this->Book.begin(), this->Book.end(), [](const CContact& lhs, const CContact& rhs){
00189 //        return pr(lhs, rhs);
00190 //    });
00191     std::sort(this->Book.begin(), this->Book.end(), [](const CContact& lhs, const CContact& rhs){
00192         return CContact::prSortByGroup(lhs, rhs);
00193     });
00194 }
00195
00218 void AddressBook::List()
00219 {
00224     for (std::vector<CContact>::iterator it = this->Book.begin(); it != this->Book.end(); ++it)
00225     { // can also be it++
00226         std::cout << *it;
00227     }
00228 }
00229
00235 void AddressBook::ListGroup(std::string &group)
00236 {
00237     for (std::vector<CContact>::iterator iteratorOfContact = this->Book.begin(); iteratorOfContact !=
    this->Book.end(); iteratorOfContact++)
00238     {
00239         std::string name = "", phoneNumber = "";
```

```
00240         if ((*iteratorOfContact).PatternMatch(name, phoneNumber, group))
00241         {
00242             std::cout << *iteratorOfContact;
00243         }
00244     }
00245 }
```

10.3 AddressBook.h 文件参考

```
#include "CContact.h"
#include <QSqlQueryModel>
#include <string>
#include <vector>
#include <algorithm>
#include <iostream>
```

类

- class AddressBook
  地址簿类

10.4 AddressBook.h

浏览该文件的文档.

```
00001 /*
00002 * @Author:  Frank Chu
00003 * @Date:  2022-11-16 16:04:46
00004 * @LastEditors:  Frank Chu
00005 * @LastEditTime:  2022-12-15 16:29:43
00006 * @FilePath:  /Cpp/lab/Cpp-lab01-week11/source/AddressBook.h
00007 * @Description:
00008 *
00009 * Copyright (c) 2022 by Frank Chu, All Rights Reserved.
00010 */
00011
00012 #ifndef ADDRESSBOOK_H
00013 #define ADDRESSBOOK_H
00014
00015 #include "CContact.h"
00016 #include <QSqlQueryModel>
00017
00018 #include <string>
00019 #include <vector>
00020 #include <algorithm>
00021 #include <iostream>
00022
00029 class AddressBook
00030 {
00031 private:
00035     std::vector<CContact> Book;
00036     int id;
00037
00038 public:
00039
00043     AddressBook();
00044
00048     ~AddressBook();
00049
00055     void setId(int id) { this->id = id; }
00056
00061     int getId()const { return this->id; }
00062
00069     void AddContact(std::string &, std::string &, std::string &);
00070
00075     void Add(CContact& contact);
00076
```

```
00082     CContact operator[](int indexOfContact);
00083
00087     void Sort();
00088
00092     void SortGroup();
00093
00097     void List();
00098
00103     void ListGroup(std::string& group);
00104
00112     int Delete(std::string& Name, std::string& Number, std::string& Group); //按条件删除联系人，返回删除的
      人数。如果没有删除任何人，返回0
00113
00122     int Find(int startIndex, std::string& name, std::string& number, std::string& group);
00123
00130     bool indexSafe(int index);
00131 };
00132
00133 #endif // ADDRESSBOOK_H
```

10.5 CategoryOfAddressBook.cpp 文件参考

```
#include <QString>
```

类

- struct Category

10.6 CategoryOfAddressBook.cpp

浏览该文件的文档.

```
00001 #include <QString>
00002
00003 struct Category{
00004     QString friends = "Friends";
00005     QString colleague = "Colleague";
00006     QString family = "Family";
00007     QString relative = "Relatives";
00008     QString student = "Students";
00009
00010     QString name = "Name";
00011     QString phoneNumber = "Phone Numebr";
00012     QString group = "Group";
00013 };
00014
```

10.7 CContact.cpp 文件参考

```
#include "CContact.h"
```

函数

- std::ostream & operator<< (std::ostream &os, CContact contactInfo)
- std::istream & operator>> (std::istream &is, CContact &contactToBeRevised)
- bool match (std::string &pattern, std::string &source)

  字符串匹配，判断字符串*source*是否匹配*pattern*，或者说字符串*source*是*pattern*所表达的集合中的某个成员
- bool pr (const CContact &lhsContact, const CContact &rhsContact)

10.7.1 函数说明

10.7.1.1 match()

```
bool match (
              std::string & pattern,
              std::string & source )
```

字符串匹配，判断字符串source是否匹配pattern，或者说字符串source是pattern所表达的集合中的某个成员

`string::npos` 静态成员常量

是对类型为 `size_t` 的元素具有最大可能的值。 当这个值在字符串成员函数中的长度或者子长度被使用时，该值表示“直到字符串结尾”。 作为返回值他通常被用作表明没有匹配。

```
if (s1.find(s2) != std::string::npos) {
    std::cout << "found!" << '\n';
}
```

测试

```
std::string matchTestCase1Pattern = "Franek", matchTestCase1Source = "Frank Chu";
if(match(matchTestCase1Pattern, matchTestCase1Source)) {
    std::cout << "Yes" << "\n";
} else {
    std::cout << "No" << "\n";
}
```

参见

- Check if a string contains a string in C++ https://stackoverflow.com/questions/2340281/check-if-
- C++ 中 `string::find()` 函数和 `string::npos` 函数的使用 https://www.cnblogs.com/lixuejian/p/10844905.html
- std::string::find 空字符串返回结果不是 string::npos https://blog.csdn.net/yasi_xi/article/details/7305443
- 查找字符串，支持通配符查找，通配符包含 .和? https://blog.csdn.net/wang_anna/article/details/117019969
- 通配符（?，*）与正则表达式 https://blog.csdn.net/yh13572438258/article/details/12154
- str::string和wchar_t*相互转化 https://blog.csdn.net/zddblog/article/details/38670349
- C++：wchar_t* & string相互转换 https://codeantenna.com/a/uDA7bfXIkF
- C++11之正则表达式（regex_match、regex_search、regex_replace） https://blog.csdn.net/qq_45254369/article/details/125491031

不区分大小写，需包含头文件 From: 查找字符串，支持通配符查找，通配符包含 .和? https://blog.csdn.net/wang_anna/article/details/117019969

```
#include<regex>
using namespace regex_constants;
ECMAScript | icase // Case insensitive
```

在文件 CContact.cpp 第 146 行定义.

```
00146                                                {
00147     auto positionOfQuestionMark = pattern.find("?");
00148     auto positionOfAsteriskMark = pattern.find("*");
00149     if(positionOfQuestionMark != std::string::npos) {
00150         pattern.replace(positionOfQuestionMark, 1, ".");
00151     }
00152     if(positionOfAsteriskMark != std::string::npos) {
00153         pattern.replace(positionOfAsteriskMark, 1, ".*");
00154     }
00155
00156     return std::regex_match(source, std::regex(pattern));
00157     // return (source.find(pattern) != std::string::npos) ?  true :  false;
00158 }
```

10.7.1.2 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            CContact contactInfo )
```

参见

【懒猫老师-最简版C++-(18)类的友元】 https://www.bilibili.com/video/BV127411↩ Q7eu/

注解

Overloading the << Operator for Your Own Classes https://learn.microsoft.com/en-us/cpp/standard-li

在文件 CContact.cpp 第 70 行定义.

```
00070                                                   {
00071     // Overloading the >> Operator for Your Own Classes
00072     os << contactInfo.Name << ", " << contactInfo.Number << ", " << contactInfo.Group << "\n";
00073     return os;
00074 }
```

10.7.1.3 operator>>()

```
std::istream & operator>> (
            std::istream & is,
            CContact & contactToBeRevised )
```

注解

Overloading the >> Operator for Your Own Classes https://learn.microsoft.com/en-us/cpp/standard-li

在文件 CContact.cpp 第 77 行定义.

```
00077                                                   {
00078     std::string name, number, group;
00079     is >> name >> number >> group;
00080     contactToBeRevised.setContact(name, number, group);
00081     // std::cout << name << number << group;
00082     return is;
00083 }
```

10.7.1.4 pr()

```
bool pr (
            const CContact & lhsContact,
            const CContact & rhsContact )
```

参数

lhsContact	left hand side of Contact to be compared
rhsContact	right hand side of Contact to be compared

返回

true lhsContact.Group < rhsContact.Group;

false lhsContact.Group > rhsContact.Group;

测试

```
std::string name = "Frank Chu", number = "1596", group = "Student";
std::string rhsName = "Panda", rhsNumber = "22", rhsGroup = "Teacher";
if(pr(CContact(name, number, group), CContact(rhsName, rhsNumber, rhsGroup))) {
    std::cout << "< change postion" << "\n";
} else {
    std::cout << "> do not change" << "\n";
}
```

在文件 CContact.cpp 第 160 行定义.

```
00160                                                                    {
00161     return lhsContact.Group < rhsContact.Group;
00162 }
```

10.8 CContact.cpp

浏览该文件的文档.

```
00001 /*
00002 * @Author:  Frank Chu
00003 * @Date:  2022-11-16 16:25:59
00004 * @LastEditors:  Frank Chu
00005 * @LastEditTime:  2022-11-21 22:10:57
00006 * @FilePath:  /Cpp/lab/Cpp-lab01-week11/source/CContact.cpp
00007 * @Description:
00008 *
00009 * Copyright (c) 2022 by Frank Chu, All Rights Reserved.
00010 */
00011
00012 #include "CContact.h"
00013
00014 CContact::CContact() {}
00015
00017 CContact::CContact(std::string& Name, std::string& Number, std::string& Group) {
00018     this->Name = Name;
00019     this->Number = Number;
00020     this->Group = Group;
00021 }
00022
00023 CContact::CContact(const CContact& ContactInfo) {
00024     this->Name = ContactInfo.Name;
00025     this->Number = ContactInfo.Number;
00026     this->Group = ContactInfo.Group;
00027 }
00028
00029 CContact::~CContact() { }
00030
00031
00050 bool CContact::operator<(const CContact& contactToBeCompared)const {
00051     return this->Name < contactToBeCompared.Name;
00052     // return (this->Name < contactToBeCompared.Name) ?  true :  false;
00053     // if (Name < contactToBeCompared.Name) {
00054     //     return true;
00055     // } else {
00056     //     return false;
00057     // }
00058 }
00059
00061 CContact& CContact::operator=(const CContact& oldContact) {
00062     this->Name = oldContact.Name;
00063     this->Number = oldContact.Number;
00064     this->Group = oldContact.Group;
00065     return *this;
00066 }
00067
00070 std::ostream &operator<<(std::ostream& os, CContact contactInfo) {
00071     // Overloading the >> Operator for Your Own Classes
00072     os << contactInfo.Name << ", " << contactInfo.Number << ", " << contactInfo.Group << "\n";
00073     return os;
00074 }
00075
00077 std::istream &operator>>(std::istream& is, CContact& contactToBeRevised) {
```

```
00078     std::string name, number, group;
00079     is >> name >> number >> group;
00080     contactToBeRevised.setContact(name, number, group);
00081     // std::cout << name << number << group;
00082     return is;
00083 }
00084
00085 void CContact::getContact(std::string& Name, std::string& Number, std::string& Group) {
00086     Name = this->Name;
00087     Number = this->Number;
00088     Group = this->Group;
00089 }
00090
00091 void CContact::setContact(std::string& Name, std::string& Number, std::string& Group) {
00092     this->Name = Name;
00093     this->Number = Number;
00094     this->Group = Group;
00095 }
00096
00101 bool CContact::PatternMatch(std::string& NamePattern, std::string& NumberPattern, std::string&
      GroupPattern) {
00102     return (
00103         match(NamePattern, this->Name) && match(NumberPattern, this->Number) && match(GroupPattern,
      this->Group)
00104     ) ?  true :  false;
00105 }
00106
00146 bool match(std::string &pattern, std::string &source) {
00147     auto positionOfQuestionMark = pattern.find("?");
00148     auto positionOfAsteriskMark = pattern.find("*");
00149     if(positionOfQuestionMark != std::string::npos) {
00150         pattern.replace(positionOfQuestionMark, 1, ".");
00151     }
00152     if(positionOfAsteriskMark != std::string::npos) {
00153         pattern.replace(positionOfAsteriskMark, 1, ".*");
00154     }
00155
00156     return std::regex_match(source, std::regex(pattern));
00157     // return (source.find(pattern) != std::string::npos) ?  true :  false;
00158 }
00159
00160 bool pr (const CContact& lhsContact, const CContact& rhsContact) {
00161     return lhsContact.Group < rhsContact.Group;
00162 }
00163
00164 /*
00165 0xFFFF搬砖艺术
00166 茶黑 23:19:25
00167 doxygen 有什么推荐教程不，就是那个写注释的，想在 vs code 里面用
00168 还有就是 vs code 自带的这个注释工具只支持一部分的语法吗，这个有文档不，查了一下没找到[表情]
00169
00170 @茶黑 你可以写个代码片段按doxygen规则写一个就行
00171
00172 @0x1234
      https://devblogs.microsoft.com/cppblog/visual-studio-code-c-extension-july-2020-update-doxygen-comments-and-logpoints/
00173 茶黑 00:37:25
00174 我看了下，好像就提到的几个 doxygen 标签可以正常用，其他官方的 intelliSense 没支持好像。cpp 这个注释有点令人难受。隔壁
      js 可以直接写 example code，复制粘贴
00175
00176 @0x1234 我反正试了好久，查了好久，各种翻 github issue，目前看来是这样的。如果查到其他资料，请 at 我[表情]
00177
00178 还有很多奇葩问题，换行只能识别成空格这些
00179 https://github.com/microsoft/vscode-cpptools/issues/5741
00180 the missing newline becomes a space
00181
00182 设置不简化注释，但也有问题
00183 https://github.com/microsoft/vscode-cpptools/issues/8525
00184 If true, tooltips of hover and auto-complete will only display certain labels of structured comments.
      Otherwise, all
00185 comments are displayed.
00186
00187
      https://devblogs.microsoft.com/cppblog/visual-studio-code-c-extension-july-2020-update-doxygen-comments-and-logpoints/
00188 Visual Studio Code C++ Extension July 2020 Update:  Doxygen comments and Log points
00189
00190 C/C++ doc comments preview doesn't react to newlines #3464
00191 https://github.com/microsoft/vscode-cpptools/issues/3464
00192 */
```

10.9 CContact.h 文件参考

```
#include <QUuid>
#include <iostream>
#include <string>
#include <regex>
```

类

- class CContact

  CContact 是联系人类

函数

- bool match (std::string &pattern, std::string &source)

  字符串匹配，判断字符串*source*是否匹配*pattern*，或者说字符串*source*是*pattern*所表达的集合中的某个成员

10.9.1 函数说明

10.9.1.1 match()

```
bool match (
            std::string & pattern,
            std::string & source )
```

字符串匹配，判断字符串source是否匹配pattern，或者说字符串source是pattern所表达的集合中的某个成员

参数

pattern	Pattern string including '*'
source	Source string

返回

true can find pattern in the source

false cannot find pattern in the source

`string::npos` 静态成员常量

是对类型为 `size_t` 的元素具有最大可能的值。 当这个值在字符串成员函数中的长度或者子长度被使用时，该值表示“直到字符串结尾”。 作为返回值他通常被用作表明没有匹配。

```
if (s1.find(s2) != std::string::npos) {
    std::cout << "found!" << '\n';
}
```

测试

```
std::string matchTestCase1Pattern = "Franek", matchTestCase1Source = "Frank Chu";
if(match(matchTestCase1Pattern, matchTestCase1Source)) {
    std::cout << "Yes" << "\n";
} else {
    std::cout << "No" << "\n";
}
```

参见

- Check if a string contains a string in C++ https://stackoverflow.com/questions/2340281/check-if-
- C++ 中 string::find() 函数和 string::npos 函数的使用 https://www.cnblogs.com/lixuejian/p/10844905.html
- std::string::find 空字符串 返回结果不是 string::npos https://blog.csdn.net/yasi_xi/article/details/7305443
- 查找字符串，支持通配符查找，通配符包含 .和? https://blog.csdn.net/wang_anna/article/details/117019969
- 通配符（?，*）与正则表达式 https://blog.csdn.net/yh13572438258/article/details/12154
- str::string和wchar_t*相互转化 https://blog.csdn.net/zddblog/article/details/38670349
- C++：wchar_t* & string相互转换 https://codeantenna.com/a/uDA7bfXIkF
- C++11之正则表达式（regex_match、regex_search、regex_replace） https://blog.csdn.net/qq_45254369/article/details/125491031

不区分大小写，需包含头文件 From: 查找字符串，支持通配符查找，通配符包含 .和? https://blog.csdn.net/wang_anna/article/details/117019969

```
#include<regex>
using namespace regex_constants;
ECMAScript | icase // Case insensitive
```

在文件 CContact.cpp 第 146 行定义.

```
00146                                                {
00147     auto positionOfQuestionMark = pattern.find("?");
00148     auto positionOfAsteriskMark = pattern.find("*");
00149     if(positionOfQuestionMark != std::string::npos) {
00150         pattern.replace(positionOfQuestionMark, 1, ".");
00151     }
00152     if(positionOfAsteriskMark != std::string::npos) {
00153         pattern.replace(positionOfAsteriskMark, 1, ".*");
00154     }
00155 
00156     return std::regex_match(source, std::regex(pattern));
00157     // return (source.find(pattern) != std::string::npos) ?  true :  false;
00158 }
```

10.10 CContact.h

浏览该文件的文档.

```
00001 /*
00002 * @Author:  Frank Chu
00003 * @Date:  2022-11-16 13:11:56
00004 * @LastEditors:  Frank Chu
00005 * @LastEditTime:  2022-11-22 00:50:06
00006 * @FilePath:  /Cpp/lab/Cpp-lab01-week11/source/CContact.h
00007 * @Description:
00008 *
00009 * Copyright (c) 2022 by Frank Chu, All Rights Reserved.
00010 */
00011 
00012 #ifndef CCONTACT_H
00013 #define CCONTACT_H
00014 
00015 #include <QUuid>
00016 
00017 #include <iostream>
00018 #include <string>
```

```
00019 #include <regex>
00020
00027 class CContact
00028 {
00029 private:
00030     std::string Name;
00031     std::string Number;
00032     std::string Group;
00033     QUuid id;
00034 public:
00038     CContact();
00039
00044     CContact(std::string &, std::string &, std::string &);
00045
00048     CContact(const CContact &);
00049
00051     virtual ~CContact();
00052
00057     void getContact(std::string &, std::string &, std::string &);
00058
00063     void setContact(std::string &, std::string &, std::string &);
00064
00071     bool operator<(const CContact &) const;
00072
00075     CContact& operator=(const CContact &);
00076
00081     friend std::ostream &operator<<(std::ostream &, CContact);
00082
00087     friend std::istream &operator>>(std::istream &, CContact &);
00088
00106     friend bool pr (const CContact& lhsContact, const CContact& rhsContact);
00107
00108     static bool prSortByGroup(const CContact& lhsContact, const CContact& rhsContact) {
00109         return lhsContact.Group < rhsContact.Group;
00110     }
00111
00112
00113
00122     bool PatternMatch(std::string& name, std::string& number, std::string& group);
00123
00124 };
00125
00126
00135 bool match(std::string &pattern, std::string &source);
00136
00137 #endif // CCONTACT_H
00138
```

10.11 main.cpp 文件参考

```
#include "widget.h"
#include <QApplication>
#include <vector>
```

函数

- int main (int argc, char ∗argv[ ])

10.11.1 函数说明

10.11.1.1 main()

```
int main (
              int argc,
              char * argv[] )
```

在文件 main.cpp 第 6 行定义.

```
00007 {
00008     QApplication a(argc, argv);
00009     Widget w;
00010     w.show();
00011     return a.exec();
00012 }
```

10.12 main.cpp

浏览该文件的文档.

```
00001 #include "widget.h"
00002
00003 #include <QApplication>
00004 #include <vector>
00005
00006 int main(int argc, char *argv[])
00007 {
00008     QApplication a(argc, argv);
00009     Widget w;
00010     w.show();
00011     return a.exec();
00012 }
```

10.13 README.md 文件参考

10.14 widget.cpp 文件参考

```
#include "widget.h"
#include "ui_widget.h"
```

10.15 widget.cpp

浏览该文件的文档.

```
00001 #include "widget.h"
00002 #include "ui_widget.h"
00003
00014 Widget::Widget(QWidget *parent) :  QWidget(parent) ,
00015     ui(new Ui::Widget)
00016 {
00017     this->ui->setupUi(this);
00018     this->Bind();
00019     this->fillComboBoxAndTableWidget();
00020 //    this->addTemplateData();
00021 }
00022
00023 Widget::~Widget()
00024 {
00025     delete this->ui;
00026 }
00027
00028 void Widget::Bind() {
00029     QObject::connect(ui->nameLineEdit, SIGNAL(returnPressed()), this, SLOT(ExitWidget()));
00030     QObject::connect(ui->exitButton, SIGNAL(clicked(bool)), this, SLOT(ExitWidget()));
```

```
00031 //    QObject::connect(ui->exitButton, &QPushButton::clicked, this, &Widget::ExitWidget);
00032 //    QObject::connect(ui->exitButton, SLGNAL(clicked()), this, SLOT(ExitWidget()));
00033 //    connect(ui->loadButton, &QPushButton::clicked, [this](){
00034 //        QMessageBox::information(this, "Info", "Click to explore");
00035 //    });
00036
00037 //    QObject::connect(ui->loadButton, SIGNAL(clicked()), this, SLOT(LoadFile()));
00038
00039     QObject::connect(this->ui->newButton, &QPushButton::clicked, this, &Widget::newButtonSlot);
00040     QObject::connect(this->ui->loadButton, &QPushButton::clicked, this, &Widget::LoadFile);
00041     QObject::connect(this->ui->saveButton, &QPushButton::clicked, this, &Widget::saveButtonSlot);
00042     QObject::connect(this->ui->tableView, &QTableView::clicked, this, &Widget::clickedTableView);
00043     QObject::connect(this->ui->deleteButton, &QPushButton::clicked, this, &Widget::deleteARow);
00044     QObject::connect(this->ui->updateButton, &QPushButton::clicked, this, &Widget::alterALine);
00045
00046 }
00047
00064 void Widget::fillComboBoxAndTableWidget()
00065 {
00066
00067 //    this->ui->groupComboBox->setModel();
00068 //    this->ui->groupComboBox->setModel();
00069     this->addressBook.setId(0);
00070
00071     // ComboBox
00072     QStringList groupItems;
00073     groupItems.append(categories.friends);
00074     groupItems.append(categories.colleague);
00075     groupItems.append(categories.family);
00076     groupItems.append(categories.relative);
00077     groupItems.append(categories.student);
00078     this->ui->groupComboBox->addItems(groupItems);
00079
00080     this->ui->groupComboBox->setCurrentText(categories.student);
00081
00082 //    qDebug() << this->ui->groupComboBox->currentText();
00083
00084     // QTableWidget 小部件的使用
00085     QStringList header;
00086     header << categories.name << categories.phoneNumber << categories.group;
00087     this->ui->tableWidget->setColumnCount(3); //设定列总数为  3
00088     this->ui->tableWidget->setHorizontalHeaderLabels(header); //设定列标题
00089
00090     // Table View(Model Based) Title
00091     this->model = new QStandardItemModel;
00092     this->model->setHorizontalHeaderLabels(header);
00093
00094     // Table View(Model Based) model for view
00095     this->ui->tableView->setModel(model);
00096     this->ui->tableWidget->hide();
00097 //    this->ui->tableView->hide();
00098
00099     this->customizedTableStyle();
00100 }
00101
00102 void Widget::newButtonSlot() {
00103 //    this->ui->textEdit->clear();
00104 //    this->setWindowTitle("New Address Book");
00105 //    QUuid id = QUuid();
00106
00107     // 获得行计数
00108 //    int lastRow = this->model->rowCount();
00109 //    qDebug() << lastRow;
00110
00111     std::string name = this->ui->nameLineEdit->text().toStdString();
00112     std::string phone = this->ui->phoneLineEdit->text().toStdString();
00113     std::string group = this->ui->groupComboBox->currentText().toStdString();
00114 //    qDebug() << id << QString::fromStdString(name) << this->ui->groupComboBox->currentText();
00115
00116     this->addressBook.AddContact(name, phone, group);
00117     QStandardItem* nameItem = new QStandardItem(QString::fromStdString(name));
00118     QStandardItem* phoneItem = new QStandardItem(QString::fromStdString(phone));
00119     QStandardItem* groupItem = new QStandardItem(QString::fromStdString(group));
00120
00121     QList<QStandardItem*> itemList;
00122     itemList << nameItem << phoneItem << groupItem;
00123
00124     // 在表格后添加一行
00125     model->appendRow(itemList);
00126
00127     this->clearLineEdit();
00128 }
00129
00134 void Widget::LoadFile() {
00135
00136     // 获取应用程序的路径
00137     // 调用打开文件对话框打开一个文件
```

```
00138     QString fileName = QFileDialog::getOpenFileName(this, "Choose a file",
      QApplication::applicationDirPath(),
00139                                                     "Text File(*.txt);;All File(*.*)");
00140 //    QString fileName = QFileDialog::getOpenFileName(this, "Choose a File",
00141 //                                                    QCoreApplication::applicationFilePath());
00142 //    QFileDialog::getOpenFileName(this, "Choose a File", QCoreApplication::applicationDirPath());
00143
00144     // 如果未选择文件，退出
00145     if (fileName.isEmpty()) {
00146         QMessageBox::warning(this, "Warning", "Choose a File, please");
00147         return;
00148     }
00149 //    qDebug() << fileName;
00150
00151     // 文件内容字符串列表
00152     QStringList fileContentStringList;
00153
00154     // Create a QFile Object
00155     // https://doc.qt.io/qt-6/qfile.html
00156     // 以文件方式读出
00157     QFile file(fileName);
00158
00159     // 以只读文本方式打开文件
00160     file.open(QIODevice::ReadOnly);
00161
00162     // 用文本流读取文件
00163     QTextStream fileStream(&file);
00164
00165     // 清空
00166     this->ui->textEdit->clear();
00167
00168     while(!fileStream.atEnd()) {
00169         // 读取文件的一行
00170         QString stringToBeDisplayed = fileStream.readLine();
00171
00172         // 添加到文本框显示
00173         this->ui->textEdit->append(stringToBeDisplayed);
00174
00175         // 添加到 fileContentStringList
00176         fileContentStringList.append(stringToBeDisplayed);
00177     }
00178
00179     // 关闭文件
00180     //    QByteArray byteArray = file.readAll();
00181     //    ui->textEdit->setText(QString(byteArray));
00182     file.close();
00183
00184     // 窗口标题显示
00185     this->setWindowTitle(fileName);
00186
00187     // 从StringList的内容初始化数据模型
00188     this->initModelFromStringList(fileContentStringList);
00189 }
00190
00195 void Widget::saveButtonSlot() {
00196
00197     // 获取应用程序的路径
00198 //    QString curPath = QCoreApplication::applicationDirPath();
00199 //    qDebug() << curPath;
00200
00201     // Save New Dialog
00202     // 调用打开文件对话框选择一个文件
00203     QString fileName = QFileDialog::getSaveFileName(this, "Chose a file",
      QCoreApplication::applicationFilePath(),
00204                                                     "Text File(*.txt);;All File(*.*)");
00205
00206
00207     // 未选择文件，退出
00208     if (fileName.isEmpty()) {
00209         QMessageBox::warning(this, "Warning", "Choose a file");
00210         return;
00211     }
00212
00213     QFile file(fileName);
00214     file.open(QIODevice::WriteOnly);
00215
00216 //    QByteArray ba;
00217 //    ba.append(this->ui->textEdit->toPlainText().toUtf8());
00218 //    file.write(ba);
00219 //    file.close();
00220
00221     QString stringToBeStored;
00222
00223     // 用文本流读取文件
00224     QTextStream fileStream(&file);
00225     QStandardItem* item;
00226
```

```
00227     // 获取表头文字
00228     for(int i = 0; i < this->model->columnCount(); i++) {
00229
00230         // 获取表头的项数据
00231         item = this->model->horizontalHeaderItem(i);
00232
00233         // 以 TAB 间隔开
00234         stringToBeStored += item->text() + "\t\t";
00235     }
00236
00237     // 文件里需要加入换行符 \n
00238     fileStream << stringToBeStored << "\n";
00239
00240     this->ui->textEdit->append(stringToBeStored);
00241
00242     // 获取数据区文字
00243     for (int i = 0; i < this->model->rowCount(); i++) {
00244         stringToBeStored = "";
00245         for (int j = 0; j < this->model->columnCount(); j++) {
00246             item = this->model->item(i, j);
00247             stringToBeStored += item->text() + "\t\t";
00248         }
00249
00250         this->ui->textEdit->append(stringToBeStored);
00251         fileStream << stringToBeStored << "\n";
00252     }
00253
00254     file.close();
00255 }
00256
00260 void Widget::ExitWidget()
00261 {
00262 //    QString program = "/System/Applications/Utilities/Terminal.app";
00263 //    program = "/Applications/" + this->ui->nameLineEdit->text() + ".app";
00264 //    this->ui->nameLineEdit->setText(program);
00265 //    QProcess *myProcess = new QProcess(this);
00266 //    myProcess->start(program);
00267     this->close();
00268 }
00269
00270 void Widget::keyPressEvent(QKeyEvent *key) {
00271     if(key->modifiers() == Qt::ControlModifier && key->key() == Qt::Key_S) {
00272         this->saveButtonSlot();
00273     }
00274     if(key->modifiers() == Qt::ControlModifier && key->key() == Qt::Key_O) {
00275         this->LoadFile();
00276     }
00277 }
00278
00279 void Widget::addTemplateData()
00280 {
00281     QStandardItem* name = new QStandardItem(QString("apple"));
00282     QStandardItem* phone = new QStandardItem(QString("400-666-8800"));
00283     QStandardItem* group = new QStandardItem(QString("Colleague"));
00284
00285     QList<QStandardItem*> itemList;
00286     itemList << name << phone << group;
00287     model->appendRow(itemList);
00288
00289     name = new QStandardItem(QString("google"));
00290     phone = new QStandardItem(QString("901-3283-2233"));
00291     group = new QStandardItem(QString("Colleague"));
00292     itemList.clear();
00293     itemList << name << phone << group;
00294     model->appendRow(itemList);
00295 }
00296
00297 void Widget::clearLineEdit()
00298 {
00299     // clear Text
00300     this->ui->nameLineEdit->setText("");
00301     this->ui->phoneLineEdit->setText("");
00302 }
00303
00322 void Widget::initModelFromStringList(QStringList& fileContentStringList)
00323 {
00324     // 将内存中的 model 数据模型清除
00325     this->model->clear();
00326
00327     // 从一个StringList 获取数据，初始化数据Model
00328     // 文本行数，第1行是标题
00329     int rowCount = fileContentStringList.count();
00330
00331     // 实际数据行数
00332     this->model->setRowCount(rowCount - 1);
00333
00334     // 设置表头
```

```
00335     // 第1行是表头
00336     QString header = fileContentStringList.at(0);
00337
00349     static auto multiTab = QRegularExpression("\\t+");
00350
00351     // 一个或多个空格、TAB("\s+") 多个 TAB 分隔("\t+")等分隔符隔开的字符串， 分解为一个StringList
00352     QStringList headerList = header.split(multiTab);
00353     headerList.removeLast();
00354     //设置表头文字
00355     this->model->setHorizontalHeaderLabels(headerList);
00356
00357     //设置表格数据
00358 //    QString lineText;
00359 //    QStringList lineTempList;
00360     QStandardItem *itemStandard;
00361     for(int i = 0; i < rowCount; i++) {
00362
00363         //获取数据区的一行
00364         QString lineText = fileContentStringList.at(i);
00365
00366         //一个或多个空格、TAB等分隔符隔开的字符串， 分解为一个StringList
00367         QStringList lineTempList = lineText.split(multiTab);
00368
00369         //lineTempList 的行数等于 headerList.count, 固定的
00370         for(int j = 0; j < headerList.count(); j++) {
00371
00372             //创建item
00373             itemStandard = new QStandardItem(lineTempList.at(j));
00374
00375             //为模型的某个行列位置设置Item
00376             this->model->setItem(i - 1, j, itemStandard);
00377         }
00378     }
00379     this->customizedTableStyle();
00380 }
00381
00382 void Widget::customizedTableStyle()
00383 {
00384     // Style sheet
00385     //设定列延展特性
00386     this->ui->tableWidget->horizontalHeader()->setStretchLastSection(true);
00387     this->ui->tableView->horizontalHeader()->setStretchLastSection(true);
00388
00389     // QTableView column width
00390     // https://stackoverflow.com/questions/26681578/qtableview-column-width
00391     this->ui->tableView->setColumnWidth(1, 200);
00392
00393     //设定标题样式
00394     this->ui->tableWidget->horizontalHeader()->setStyleSheet("border:none; border-bottom:1px solid
      grey");
00395     this->ui->tableView->horizontalHeader()->setStyleSheet("border:none; border-bottom:1px solid
      grey");
00396
00397     //设定表格行选定行为
00398     this->ui->tableWidget->setSelectionMode(QAbstractItemView::SingleSelection);
00399     this->ui->tableWidget->setSelectionBehavior(QAbstractItemView::SelectRows);
00400
00401     this->ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
00402
00403     //显示网格线
00404     this->ui->tableWidget->showGrid();
00405     this->ui->tableView->showGrid();
00406 }
00407
00408 void Widget::clickedTableView()
00409 {
00410     QAbstractItemModel* modelSelected = this->ui->tableView->model();
00411     int row = this->ui->tableView->currentIndex().row();
00412 //    qDebug() << row;
00413
00414     //根据行号在地址簿中读取并显示相应的联系人至编辑界面
00415     this->ui->nameLineEdit->setText(modelSelected->index(row, 0).data().toString());
00416     this->ui->phoneLineEdit->setText(modelSelected->index(row, 1).data().toString());
00417     this->ui->groupComboBox->setCurrentText(modelSelected->index(row, 2).data().toString());
00418
00419     //选中一行后启用按钮
00420     this->ui->deleteButton->setEnabled(true);
00421     this->ui->updateButton->setEnabled(true);
00422 }
00423
00427 void Widget::deleteARow()
00428 {
00429     // 获取当前联系人行
00430     auto currentRow = this->ui->tableView->currentIndex().row();
00431
00432     // 将联系人从数据模型中删除
00433     this->model->removeRow(currentRow);
```

```
00434
00435     // 调整相关按钮 LineEdit 状态
00436     this->clearLineEdit();
00437     this->ui->deleteButton->setEnabled(false);
00438     this->ui->updateButton->setEnabled(false);
00439 }
00440
00441 void Widget::alterALine()
00442 {
00443     // 将数据先保存到地址簿当下位置
00444     // 获取当前联系人行
00445     this->newButtonSlot();
00446     this->deleteARow();
00447 }
```

10.16 widget.h 文件参考

```
#include "CategoryOfAddressBook.cpp"
#include "AddressBook.h"
#include <QWidget>
#include <QProcess>
#include <QMessageBox>
#include <QFileDialog>
#include <QDebug>
#include <QByteArray>
#include <QKeyEvent>
#include <QStandardItemModel>
#include <QRegularExpression>
```

类

- class Widget

命名空间

- namespace Ui

10.17 widget.h

浏览该文件的文档.

```
00001 #ifndef WIDGET_H
00002 #define WIDGET_H
00003 #include "CategoryOfAddressBook.cpp"
00004 #include "AddressBook.h"
00005
00006 #include <QWidget>
00007 #include <QProcess>
00008 #include <QMessageBox>
00009 #include <QFileDialog>
00010 #include <QDebug>
00011 #include <QByteArray>
00012 #include <QKeyEvent>
00013 #include <QStandardItemModel>
00014 #include <QRegularExpression>
00015
00016 QT_BEGIN_NAMESPACE
00017 namespace Ui { class Widget; }
00018 QT_END_NAMESPACE
00019
00020 class Widget :  public QWidget
00021 {
```

```
00022     Q_OBJECT
00023
00024 public:
00025     Widget(QWidget *parent = nullptr);
00026     ~Widget();
00027
00028     AddressBook addressBook;
00029
00033     void Bind();
00034
00038     void fillComboBoxAndTableWidget();
00039
00044     void keyPressEvent(QKeyEvent *key);
00045
00046     void addTemplateData();
00047
00051     void clearLineEdit();
00052
00057     void initModelFromStringList(QStringList& fileContentStringList);
00058
00062     void customizedTableStyle();
00063
00064 private slots:
00065     void ExitWidget();
00066
00070     void LoadFile();
00071
00075     void saveButtonSlot();
00076
00080     void newButtonSlot();
00081
00085     void clickedTableView();
00086
00090     void deleteARow();
00091
00095     void alterALine();
00096
00097 private:
00101     Category categories;
00102
00103     Ui::Widget *ui;
00104
00110     QStandardItemModel* model;
00111 };
00112 #endif // WIDGET_H
```

Index