Chapter 1

README

1.1 实验一 Visual C++ 基于命令提示符的程序设计

1.1.1 一、 实验目的

1. 掌握基于命令提示符的人机交互界面设计；
2. 掌握类的定义和使用；
3. 掌握 vector 对象的使用方法；
4. 掌握泛型程序设计的基本方法。

1.1.2 二、实验内容

使用 STL 类模板 vecotr 实现一个“通讯录”程序，有联系人条目增加，删除，排序，查找，分组输出等功能，有一个简单的 CLI 界面。

1.1.3 三、主要程序说明

1.1.3.1 类的详细说明（如属性、方法）

- 5 类说明
- 5.1 `AddressBook` 类
  - Public 成员函数
  - Private 属性
  - 5.1.1 详细描述
  - 5.1.2 构造及析构函数说明
  - 5.1.3 成员函数说明
  - 5.1.4 类成员变量说明
- 5.2 `CContact` 类
  - Public 成员函数
  - Private 属性
  - 友元
  - 5.2.1 详细描述
  - 5.2.2 构造及析构函数说明
  - 5.2.3 成员函数说明
  - 5.2.4 友元及相关函数文档
  - 5.2.5 类成员变量说明

1.1.3.2 各方法的详细说明（方法名、参数、功能描述）

- 5.3 MainFunction 类，`main()` 函数中所用的调用对象方法的函数。
  - `MainFunction::HelpCommand()`

1.1.3.3 命令与对象方法的调用关系

- See MainFunction.cpp
- 5.3 MainFunction类 参考
  - `MainFunction::ExitCommand()`
  - `MainFunction::ListCommand()`
  - `MainFunction::ListGroupCommand()`
  - `MainFunction::SortCommand()`
  - `MainFunction::SortGroupCommand()`
  - `MainFunction::SystemPauseFunction()`

1.1.3.4 关键方法的实现（如添加、查找、删除联系人、排序等）

- Add
  - `MainFunction::AddCommand()`
  - `AddressBook::AddContact()`
- Find
  - `MainFunction::FindCommand()`
  - `AddressBook::Find()`
  - `CContact::PatternMatch()`
  - `match()`
- Delete
  - `MainFunction::DeleteCommand()`
  - `AddressBook::Delete()`

1.1.4 四、程序测试过程

- 测试列表

1.1.5 五、讨论及心得

1、实验过程中遇到的问题与解决方法

遇到了很多对于 C++ 语言特性不熟悉的部分，在经过网络上大量的查阅之后，最终解决了。

2、目前尚未解决的问题

基本上所有的函数均已实现。

ContactAddressBook

1.0.0

制作者 Frank Chu

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 **main ()**

成员 **MainFunction::DeleteCommand (AddressBook &Book)**

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

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

Chapter 3

待办事项列表

成员 **main ()**

convert commandOfInput into ENUM

Chapter 4

类索引

4.1 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 5

文件索引

5.1 文件列表

这里列出了所有文件，并附带简要说明:

Chapter 6

类说明

6.1 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 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

6.1.1 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 27 行定义.

6.1.2 构造及析构函数说明

6.1.2.1 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.

6.1.2.2 ~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.

6.1.3 成员函数说明

6.1.3.1 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.

6.1.3.2 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.

6.1.3.3 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.

函数调用图:

6.1.3.4 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.

这是这个函数的调用关系图:

6.1.3.5 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.

6.1.3.6 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 212 行定义.

这是这个函数的调用关系图:

6.1.3.7 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 229 行定义.

6.1.3.8 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.

6.1.3.9 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.

这是这个函数的调用关系图:

6.1.3.10 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.

函数调用图:

6.1.4 类成员变量说明

6.1.4.1 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 33 行定义.

该类的文档由以下文件生成:

- AddressBook.h
- AddressBook.cpp

6.2 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

Private 属性

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

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 &, const CContact &)
  定义组排序函数，供算法*sort*使用,

6.2.1 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 24 行定义.

6.2.2 构造及析构函数说明

6.2.2.1 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.

6.2.2.2 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.

6.2.2.3 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.

6.2.2.4 ~CContact()

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

析构函数

在文件 CContact.cpp 第 29 行定义.

6.2.3 成员函数说明

6.2.3.1 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 102 行定义.

6.2.3.2 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.

6.2.3.3 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.

6.2.3.4 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 118 行定义.

函数调用图:

6.2.3.5 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 108 行定义.

6.2.4 友元及相关函数文档

6.2.4.1 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.

6.2.4.2 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.

6.2.4.3 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 98 行定义.

6.2.5 类成员变量说明

6.2.5.1 Group

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

群组

在文件 CContact.h 第 29 行定义.

6.2.5.2 Name

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

姓名

在文件 CContact.h 第 27 行定义.

6.2.5.3 Number

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

电话号码

在文件 CContact.h 第 28 行定义.

该类的文档由以下文件生成:

- CContact.h
- CContact.cpp

6.3 MainFunction类 参考

解释在 main.cpp 中调用到的函数 Explanation of some functions used in main.cpp

Public 成员函数

- MainFunction ()

  MainFunction 的构造函数 *Construct a new Main Function object*
- ~MainFunction ()

  MainFunction 的析构函数 *Destroy the Main Function object*

静态 **Public** 成员函数

- static void SystemPauseFunction ()

  `system("pause")` 函数说明
- static void HelpCommand ()

  打印帮助手册 `AddressBook>help` *command, print Help Manual*
- static void AddCommand (AddressBook &Book)

  添加联系人命令 *add*命令格式*:* `AddressBook>add [name] [number] [group]` *command, add new contact to Address Book in the Command Line Interface Memory*
- static void FindCommand (AddressBook &Book)

  查找命令 *find [Name] [Number] [Group]* 按名字寻找某一节点，支持通配符 * *?* 查找
- static void DeleteCommand (AddressBook &Book)

  删除联系人命令，支持通配符 *?* * *delete [Name][Number][Group]*，按姓名删除某一节点
- static void ListCommand (AddressBook &Book)

  列出通讯录中所有联系人命令，`AddressBook>list` *command, print all contacts in the AddressBook*
- static void ListGroupCommand (AddressBook &Book)

  按组别列出通讯录中特定组别联系人
- static void SortCommand (AddressBook &Book)

  根据姓氏拼音升序排列，`AddressBook>sort` *command, sort contact in the AddressBook using string in ascii ascending order*
- static void SortGroupCommand (AddressBook &Book)

  按照组别拼音升序排序，`AddressBook>sortgroup` *command, sort contact in the AddressBook using string in ascii ascending order*
- static void ExitCommand ()

  退出程序，`AddressBook>exit` *command, exit the program*

6.3.1 详细描述

解释在 main.cpp 中调用到的函数 Explanation of some functions used in main.cpp

在文件 MainFunction.cpp 第 19 行定义.

6.3.2 构造及析构函数说明

6.3.2.1 MainFunction()

```
MainFunction::MainFunction ( )  [inline]
```

MainFunction 的构造函数 Construct a new Main Function object

在文件 MainFunction.cpp 第 26 行定义.

6.3.2.2 ~MainFunction()

```
MainFunction::~MainFunction ( )  [inline]
```

MainFunction 的析构函数 Destroy the Main Function object

在文件 MainFunction.cpp 第 31 行定义.

6.3.3 成员函数说明

6.3.3.1 AddCommand()

```
static void MainFunction::AddCommand (
            AddressBook & Book )  [inline], [static]
```

添加联系人命令 add命令格式: AddressBook>add [name] [number] [group] command, add new contact to Address Book in the Command Line Interface Memory

参数

Book	Address Book in the CLI Memory

在文件 MainFunction.cpp 第 67 行定义.

这是这个函数的调用关系图:

6.3.3.2 DeleteCommand()

```
static void MainFunction::DeleteCommand (
            AddressBook & Book )  [inline], [static]
```

删除联系人命令，支持通配符 ? ∗ delete [Name][Number][Group]，按姓名删除某一节点

参数

Book	Address Book in the CLI Memory

测试

```
- input
add frank 2 st
add frank 3 st
add frank 23 st
delete fr 2 st
- output
2 contact was deleted.
```

在文件 MainFunction.cpp 第 125 行定义.

这是这个函数的调用关系图:

6.3.3.3 ExitCommand()

```
static void MainFunction::ExitCommand ( )  [inline], [static]
```

退出程序，AddressBook>exit command, exit the program

在文件 MainFunction.cpp 第 171 行定义.

这是这个函数的调用关系图:

6.3.3.4 FindCommand()

```
static void MainFunction::FindCommand (
            AddressBook & Book )  [inline], [static]
```

查找命令 find [Name] [Number] [Group] 按名字寻找某一节点，支持通配符 * ? 查找

参数

Book	Address Book in the CLI Memory

参见

- CContact AddressBook::operator[ ](int indexOfContact)
- Count the size of the vector in C++ https://linuxhint.com/count-vector-size-c/

在文件 MainFunction.cpp 第 82 行定义.

这是这个函数的调用关系图:

6.3.3.5 HelpCommand()

```
static void MainFunction::HelpCommand ( )  [inline], [static]
```

打印帮助手册 AddressBook>help command, print Help Manual

参见

C++ Multiline String Literals https://linuxhint.com/c-multiline-string-literals/

在文件 MainFunction.cpp 第 56 行定义.

这是这个函数的调用关系图:

6.3.3.6 ListCommand()

```
static void MainFunction::ListCommand (
            AddressBook & Book )  [inline], [static]
```

列出通讯录中所有联系人命令，AddressBook>list command, print all contacts in the AddressBook

参数

Book	Address Book in the CLI Memory

在文件 MainFunction.cpp 第 135 行定义.

这是这个函数的调用关系图:

6.3.3.7 ListGroupCommand()

```
static void MainFunction::ListGroupCommand (
            AddressBook & Book )  [inline], [static]
```

按组别列出通讯录中特定组别联系人

参数

Book	Address Book in the CLI Memory

在文件 MainFunction.cpp 第 144 行定义.

函数调用图: 这是这个函数的调用关系图:

6.3.3.8 SortCommand()

```
static void MainFunction::SortCommand (
            AddressBook & Book )  [inline], [static]
```

根据姓氏拼音升序排列，AddressBook>sort command, sort contact in the AddressBook using string in ascii ascending order

参数

Book	Address Book in the CLI Memory

在文件 MainFunction.cpp 第 154 行定义.

这是这个函数的调用关系图:

6.3.3.9 SortGroupCommand()

```
static void MainFunction::SortGroupCommand (
            AddressBook & Book )  [inline], [static]
```

按照组别拼音升序排序，AddressBook>sortgroup command, sort contact in the AddressBook using string in ascii ascending order

参数

Book	Address Book in the CLI Memory

在文件 MainFunction.cpp 第 163 行定义.

函数调用图: 这是这个函数的调用关系图:

6.3.3.10 SystemPauseFunction()

```
static void MainFunction::SystemPauseFunction ( )  [inline], [static]
```

`system("pause")` 函数说明

注解

C语言中system(”pause”)是什么作用和意思 https://blog.csdn.net/haiross/article/details/450933

system 就是调用从程序中调用系统命令（和 shell 命令）。 system(”pause”)就是从程序里调用“pause”命令; 而“pause”这个系统命令的功能很简单， 就是在命令行上输出一行类似于“Press any key to exit”的字，等待用户按一个键，然后返回。
linux下运行程序出现“sh: 1: pause: not found” https://www.cnblogs.com/feifanrensheng/articles/8696.html
一般在windows平台写代码为了在终端看到运行结果，所以加入了 `system("pause");` 语句。 但是在linux下shell里pause不再是一条命令，因此会出现sh: 1: pause: not found的提示。 打开源文件删除system(”pause”);语句即可。 另外在用windows上VS编写的代码，在linux下运行必须先转码才行，否则会出现类似在ubuntu/linux系统下system是包含在stdlib.h头文件中的 要实现类似功能的方法，要么编程system(”read”); 要么不用system命令，直接getchar()；不过跟system(”pause”)的区别是，后者是按任意键继续，但是前面两种方法都是按回车键继续的。

在文件 MainFunction.cpp 第 49 行定义.

该类的文档由以下文件生成:

- MainFunction.cpp

Chapter 7

文件说明

7.1 AddressBook.cpp 文件参考

`#include "AddressBook.h"`
AddressBook.cpp 的引用(Include)关系图:

7.2 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 }
00189
00212 void AddressBook::List()
00213 {
00218     for (std::vector<CContact>::iterator it = this->Book.begin(); it != this->Book.end(); ++it)
00219     { // can also be it++
00220         std::cout << *it;
00221     }
00222 }
00223
00229 void AddressBook::ListGroup(std::string &group)
00230 {
00231     for (std::vector<CContact>::iterator iteratorOfContact = this->Book.begin(); iteratorOfContact !=
      this->Book.end(); iteratorOfContact++)
00232     {
00233         std::string name = "", phoneNumber = "";
00234         if ((*iteratorOfContact).PatternMatch(name, phoneNumber, group))
00235         {
00236             std::cout << *iteratorOfContact;
00237         }
00238     }
00239 }
```

7.3 AddressBook.h 文件参考

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

AddressBook.h 的引用(Include)关系图: 此图展示该文件直接或间接的被哪些文件引用了:

类

- class AddressBook
  地址簿类

7.4 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:  2022-11-22 00:50:52
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 #include <string>
00013 #include <vector>
00014 #include <algorithm>
00015 #include <iostream>
00016
00017 #include "CContact.h"
00018
00019 #ifndef ADDRESSBOOK_H
00020 #define ADDRESSBOOK_H
00027 class AddressBook
00028 {
00029 private:
00033     std::vector<CContact> Book;
00034
00035 public:
00036
00040     AddressBook();
00041
00045     ~AddressBook();
00046
00053     void AddContact(std::string &, std::string &, std::string &);
00054
00059     void Add(CContact& contact);
00060
00066     CContact operator[](int indexOfContact);
00067
00071     void Sort();
00072
00076     void SortGroup();
00077
00081     void List();
00082
00087     void ListGroup(std::string& group);
00088
00096     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
00097
00106     int Find(int startIndex, std::string& name, std::string& number, std::string& group);
00107
00114     bool indexSafe(int index);
00115 };
00116 #endif
```

7.5 CContact.cpp 文件参考

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

CContact.cpp 的引用(Include)关系图: 此图展示该文件直接或间接的被哪些文件引用了:

函数

- std::ostream & operator<< (std::ostream &os, CContact contactInfo)
- std::istream & operator>> (std::istream &is, CContact &contactToBeRevised)
- bool pr (const CContact &lhsContact, const CContact &rhsContact)
- 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

7.5.1 函数说明

7.5.1.1 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 163 行定义.

这是这个函数的调用关系图:

7.5.1.2 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/BV127411Q7eu/

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.

7.5.1.3 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.

7.5.1.4 pr()

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

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 98 行定义.

这是这个函数的调用关系图:

7.6 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
00098 bool pr(const CContact& lhsContact, const CContact& rhsContact) {
00099     return lhsContact.Group < rhsContact.Group;
00100 }
00101
00102 void CContact::getContact(std::string& Name, std::string& Number, std::string& Group) {
00103     Name = this->Name;
00104     Number = this->Number;
00105     Group = this->Group;
00106 }
00107
00108 void CContact::setContact(std::string& Name, std::string& Number, std::string& Group) {
00109     this->Name = Name;
00110     this->Number = Number;
00111     this->Group = Group;
00112 }
00113
00118 bool CContact::PatternMatch(std::string& NamePattern, std::string& NumberPattern, std::string&
      GroupPattern) {
00119     return (
00120         match(NamePattern, this->Name) && match(NumberPattern, this->Number) && match(GroupPattern,
      this->Group)
00121     ) ?  true :  false;
00122 }
00123
00163 bool match(std::string &pattern, std::string &source) {
00164     auto positionOfQuestionMark = pattern.find("?");
00165     auto positionOfAsteriskMark = pattern.find("*");
00166     if(positionOfQuestionMark != std::string::npos) {
00167         pattern.replace(positionOfQuestionMark, 1, ".");
00168     }
00169     if(positionOfAsteriskMark != std::string::npos) {
00170         pattern.replace(positionOfAsteriskMark, 1, ".*");
00171     }
00172
00173     return std::regex_match(source, std::regex(pattern));
00174     // return (source.find(pattern) != std::string::npos) ?  true :  false;
00175 }
00176
00177 /*
00178 0xFFFF搬砖艺术
00179 茶黑 23:19:25
00180 doxygen 有什么推荐教程不，就是那个写注释的，想在 vs code 里面用
00181 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]
00182
00183 @茶黑 你可以写个代码片段按doxygen规则写一个就行
00184
00185 @0x1234
      https://devblogs.microsoft.com/cppblog/visual-studio-code-c-extension-july-2020-update-doxygen-comments-and-logpoints/
00186 茶黑 00:37:25
00187 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
00188
00189 @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]
00190
00191 还有很多奇葩问题，换行只能识别成空格这些
00192 https://github.com/microsoft/vscode-cpptools/issues/5741
00193 the missing newline becomes a space
00194
00195 设置不简化注释，但也有问题
00196 https://github.com/microsoft/vscode-cpptools/issues/8525
00197 If true, tooltips of hover and auto-complete will only display certain labels of structured comments.
      Otherwise, all
00198 comments are displayed.
00199
00200
      https://devblogs.microsoft.com/cppblog/visual-studio-code-c-extension-july-2020-update-doxygen-comments-and-logpoints/
00201 Visual Studio Code C++ Extension July 2020 Update:  Doxygen comments and Log points
00202
00203 C/C++ doc comments preview doesn't react to newlines #3464
00204 https://github.com/microsoft/vscode-cpptools/issues/3464
00205 */
```

7.7 CContact.h 文件参考

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

CContact.h 的引用(Include)关系图: 此图展示该文件直接或间接的被哪些文件引用了:

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

7.7.1 函数说明

7.7.1.1 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

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 163 行定义.

这是这个函数的调用关系图:

7.8 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 #include <iostream>
00015 #include <string>
00016 #include <regex>
00017
00024 class CContact
00025 {
00026 private:
00027     std::string Name;
00028     std::string Number;
00029     std::string Group;
00030 public:
00034     CContact();
00035
00040     CContact(std::string &, std::string &, std::string &);
00041
```

```
00044     CContact(const CContact &);
00045
00047     virtual ~CContact();
00048
00053     void getContact(std::string &, std::string &, std::string &);
00054
00059     void setContact(std::string &, std::string &, std::string &);
00060
00067     bool operator<(const CContact &) const;
00068
00071     CContact& operator=(const CContact &);
00072
00077     friend std::ostream &operator<<(std::ostream &, CContact);
00078
00083     friend std::istream &operator>>(std::istream &, CContact &);
00084
00092     friend bool pr(const CContact &, const CContact &);
00093
00102     bool PatternMatch(std::string& name, std::string& number, std::string& group);
00103
00104 };
00105
00114 bool match(std::string &pattern, std::string &source);
00115
00116 #endif
```

7.9 main.cpp 文件参考

```
#include "MainFunction.cpp"
#include <iostream>
#include <string>
```

main.cpp 的引用(Include)关系图:

函数

- int main ()

7.9.1 函数说明

7.9.1.1 main()

```
int main ( )
```

返回

int

测试

```
add frank 110 stu
add panda 120 tea
add amy 110 tea
// test main()
int main() {
    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[2];
    // 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.Add(amy);
}
```

待办事项 convert commandOfInput into ENUM

在文件 main.cpp 第 51 行定义.

函数调用图:

7.10 main.cpp

浏览该文件的文档.

```
00001 /*
00002 * @Author:  Frank Chu
00003 * @Date:  2022-11-16 13:09:45
00004 * @LastEditors:  Frank Chu
00005 * @LastEditTime:  2022-11-21 23:07:17
00006 * @FilePath:  /Cpp/lab/Cpp-lab01-week11/source/main.cpp
00007 * @Description:
00008 * VSCodeでDoxygenのプレビュをしたい!
00009 * https://qiita.com/hakua-doublemoon/items/c328a7bf0bc7a1fbef14
00010 * Copyright (c) 2022 by Frank Chu, All Rights Reserved.
00011 */
00012
00013 #include "MainFunction.cpp"
00014 #include <iostream>
00015 #include <string>
00016
00051 int main() {
00052     // int position;
00053     AddressBook Book;
00054     std::string commandOfInput;
00055
00056     while (true)
00057     {
00058         std::cout << "AddressBook>";
00059         std::cin >> commandOfInput;
00060
00061         // Can strcmp() work with strings in c++?
      https://stackoverflow.com/questions/25360218/can-strcmp-work-with-strings-in-c
00062         if(commandOfInput.compare("help") == 0) {
00063             MainFunction::HelpCommand();
00064             continue;
00065         }
00066         if(commandOfInput.compare("add") == 0) {
00067             MainFunction::AddCommand(Book);
00068             continue;
00069         }
00070         if(commandOfInput.compare("find") == 0) {
00071             MainFunction::FindCommand(Book);
00072             continue;
00073         }
00074         if(commandOfInput.compare("delete") == 0) {
00075             MainFunction::DeleteCommand(Book);
00076             continue;
```

```
00077         }
00078         if(commandOfInput.compare("list") == 0) {
00079             MainFunction::ListCommand(Book);
00080             continue;
00081         }
00082         if(commandOfInput.compare("listgroup") == 0) {
00083             MainFunction::ListGroupCommand(Book);
00084             continue;
00085         }
00086         if(commandOfInput.compare("sort") == 0) {
00087             MainFunction::SortCommand(Book);
00088             continue;
00089         }
00090         if(commandOfInput.compare("sortgroup") == 0) {
00091             MainFunction::SortGroupCommand(Book);
00092             continue;
00093         }
00094         if(commandOfInput.compare("exit") == 0) {
00095             MainFunction::ExitCommand();
00096             break;
00097         }
00098         std::cout << "Error:  illegal command!  AddressBook>help for details\n";
00099     }
00100
00101 }
```

7.11 MainFunction.cpp 文件参考

```
#include "CContact.h"
#include "CContact.cpp"
#include "AddressBook.h"
#include "AddressBook.cpp"
```

MainFunction.cpp 的引用(Include)关系图: 此图展示该文件直接或间接的被哪些文件引用了:

类

- class MainFunction

  解释在 *main.cpp* 中调用到的函数 *Explanation of some functions used in main.cpp*

7.12 MainFunction.cpp

浏览该文件的文档.

```
00001 /*
00002 * @Author:  Frank Chu
00003 * @Date:  2022-11-20 20:02:30
00004 * @LastEditors:  Frank Chu
00005 * @LastEditTime:  2022-11-21 23:40:30
00006 * @FilePath:  /Cpp/lab/Cpp-lab01-week11/source/MainFunction.cpp
00007 * @Description:
00008 *
00009 * Copyright (c) 2022 by Frank Chu, All Rights Reserved.
00010 */
00011 #include "CContact.h"
00012 #include "CContact.cpp"
00013 #include "AddressBook.h"
00014 #include "AddressBook.cpp"
00015
00019 class MainFunction
00020 {
00021 private:
00022 public:
00026     MainFunction(){};
00027
00031     ~MainFunction(){};
00032
00049     static void SystemPauseFunction() { system("pause"); }
00050
00056     static void HelpCommand()
```

```
00057     {
00058         const std::string longString =
00059             R"(Example usage:
00060 add NAME PHONENUMBER GROUP
00061 delete NAMEPATTERN PHONENUMBER GROUPPATTERN
00062 find NAMEPATTERN PHONENUMBERPATTERN GROUPPATTERN
00063 list
00064 listgroup GROUP
00065 sort
00066 sortgroup
00067 exit
00068 )";
00069 std::cout << longString << "\n";
00070     }
00071
00076     static void AddCommand(AddressBook &Book)
00077     {
00078         std::string name, phoneNumber, group;
00079         std::cin >> name >> phoneNumber >> group;
00080         CContact contact(name, phoneNumber, group);
00081         Book.Add(contact);
00082     }
00083
00091     static void FindCommand(AddressBook &Book)
00092     {
00093         std::string name, number, group;
00094         std::cin >> name >> number >> group;
00095         int startIndex = 0;
00096         int indexOfContact = Book.Find(startIndex, name, number, group);
00097
00098         std::vector<int> contactIndex;
00099         contactIndex.push_back(indexOfContact);
00100
00101         if (indexOfContact == -1)
00102         {
00103             std::cout << "Cannot find specific contact in the address book.\n";
00104         }
00105         else
00106         {
00107             std::cout << Book[indexOfContact];
00108             while (indexOfContact != -1)
00109             {
00110                 indexOfContact = Book.Find(indexOfContact + 1, name, number, group);
00111                 contactIndex.push_back(indexOfContact);
00112                 if(indexOfContact != -1) {
00113                     std::cout << Book[indexOfContact];
00114                 }
00115             }
00116             std::cout << contactIndex.size() - 1 << " contact founded" << "\n";
00117         }
00118     }
00119
00134     static void DeleteCommand(AddressBook& Book) {
00135         std::string name, number, group;
00136         std::cin >> name >> number >> group;
00137         std::cout << Book.Delete(name, number, group) << " contact was deleted.  \n";
00138     }
00139
00144     static void ListCommand(AddressBook& Book)
00145     {
00146         Book.List();
00147     }
00148
00153     static void ListGroupCommand(AddressBook &Book) {
00154         std::string groupName;
00155         std::cin >> groupName;
00156         Book.ListGroup(groupName);
00157     }
00158
00163     static void SortCommand(AddressBook &Book)
00164     {
00165         Book.Sort();
00166     }
00167
00172     static void SortGroupCommand(AddressBook &Book)
00173     {
00174         Book.SortGroup();
00175     }
00176
00180     static void ExitCommand()
00181     {
00182     }
00183 };
```

7.13 README.md 文件参考

Index