全部代码：

（代码下面有运行截图）

#include <iostream>

#include <string>

#include <iomanip>

#define MAX 10

using namespace std;

#define OK 1

#define ERROR 0

typedef struct

{

string name;

string phone\_number;

} Node;

typedef struct List

{

int max\_str;

Node data;

struct List \*next;

} List;

List\* Get\_node()

{

List \*temp = new List;

if(!temp)

{

cout << "联系人空间创建失败，WRONG at Get\_node()" << endl;

return temp;

}

temp->next = NULL;

cout << "请输入联系人姓名：";

cin >> temp->data.name;

cout << "请输入联系人电话：";

cin >> temp->data.phone\_number;

return temp;

}

int Get\_list(List \*L) //L表示将来传入的头指针

{

int count = 0; //链表联系人个数

List \*p = L; //p表示真正拥有数据的第一个节点

List \*temp = NULL;

int max\_name = 0; //最长名字字符数

int max\_phone\_number = 0; //最长电话字符数

//录入数据

cout << "请输入联系人个数：";

cin >> count;

for(int i = 0 ; i < count ; ++i) //循环count次，以获取count组联系人信息

{

temp = Get\_node();

//更新最大字符串个数

if(i >= 1)

{

max\_name = ( max\_name > temp->data.name.length() ) ? max\_name : temp->data.name.length() ;

max\_phone\_number = ( max\_phone\_number > temp->data.phone\_number.length() ) ? max\_phone\_number : temp->data.phone\_number.length() ;

L->max\_str = (max\_name > max\_phone\_number) ? max\_name : max\_phone\_number;

}

else //该步骤只会执行一次

{

max\_name = p->data.name.length();

max\_phone\_number = p->data.phone\_number.length();

L->max\_str = (max\_name > max\_phone\_number) ? max\_name : max\_phone\_number;

}

p->next = temp; //新创建的临时节点赋值给p->next

p = p->next; //链表指针迁移到下一个节点

}

return OK;

}

int Show\_list(List \*L) //L表示将来传入的头指针

{

if(!L->next) //检查链表是否为空

{

cout << "当前系统没有数据，请先创建联系人信息" << endl;

cout << "Wrong at Show\_list()" << endl;

return ERROR;

}

List \*p = L->next;

//自适应！！！输出数据

cout << setw(L->max\_str) << "姓名\t" << setw(L->max\_str) << "电话" << endl;

while(p)

{

cout << setw(L->max\_str) << p->data.name << " |\t" << setw(L->max\_str) << p->data.phone\_number << endl;

p = p->next;

}

return OK;

}

int Out\_node(List \*L) //取值

{

if(!L->next) //检查链表是否为空

{

cout << "当前系统没有数据，请先创建联系人信息" << endl;

cout << "Wrong at Out\_node()" << endl;

return ERROR;

}

int index = 0; //需要取值的位置

cout << "请输入需要取值的位置：";

cin >> index;

List \*p = L;

cout << setw(L->max\_str) << "姓名\t" << setw(L->max\_str) << "电话" << endl;

for(int i = 0 ; i < index ; ++i)

{

p = p->next;

}

cout << setw(L->max\_str) << p->data.name << "|\t" << setw(L->max\_str) << p->data.phone\_number << endl;

return OK;

}

int Find\_node(List \*L) //查找

{

if(!L->next) //检查链表是否为空

{

cout << "当前系统没有数据，请先创建联系人信息" << endl;

cout << "Wrong at Find\_node()" << endl;

return ERROR;

}

string name;

cout << "请输入需要查找的联系人姓名：";

cin >> name;

List \*p = L;

while(p = p->next)

{

if(p->data.name == name)

{

cout << "电话：" << p->data.phone\_number << endl;

break;

}

}

if(!p)

{

cout << "没有找到" << name << "的电话" << endl;

}

return OK;

}

int Insert\_node(List \*L) //插入节点

{

if(!L->next) //检查链表是否为空

{

cout << "当前系统没有数据，请先创建联系人信息" << endl;

cout << "Wrong at Insert\_node()" << endl;

return ERROR;

}

int location = 0;

cout << "请输入需要插入的位置：";

cin >> location;

List \*p = L;

//遍历链表使p到达location的前一个节点

for(int i = 0 ; i < location-1 ; ++i)

{

p = p->next;

}

if(!p) //若在此处p为空了，则说明L不空，但是遍历到了最后一个节点的next，此时无法插入新节点，需要报错

{

cout << "插入位置不合法" << endl;

return ERROR;

}

else

{

List \*temp = NULL; //创建临时节点

temp = Get\_node();

//更新最大字符串个数

int max\_one = ( temp->data.name.length() > temp->data.phone\_number.length() ) ? temp->data.name.length() : temp->data.phone\_number.length();

L->max\_str = (L->max\_str > max\_one) ? L->max\_str : max\_one;

//执行插入操作

temp->next = p->next;

p->next = temp;

}

return OK;

}

int Delete\_node(List \*L)

{

if(!L->next) //检查链表是否为空

{

cout << "当前系统没有数据，请先创建联系人信息" << endl;

cout << "Wrong at Delete\_node()" << endl;

return ERROR;

}

int location = 0;

cout << "请输入需要删除的位置：";

cin >> location;

List \*p = L;

List \*temp = 0; //temp用于保存将被删除的节点，以便释放内存

for(int i = 0 ; i < location-1 ; ++i)

{

p = p->next;

}

if(!(p->next)) //若在此处p为空了，则说明L不空，但是遍历到了最后一个节点的next，此时无法插入新节点，需要报错

{

cout << "删除位置不合法" << endl;

return ERROR;

}

else

{

temp = p->next;

p->next = p->next->next;

delete temp; //释放被删除的节点的内存空间

}

return OK;

}

int Opposite\_list(List \*L)

{

if(!L->next) //检查链表是否为空

{

cout << "当前系统没有数据，请先创建联系人信息" << endl;

cout << "Wrong at Opposite\_list()" << endl;

return ERROR;

}

List \*p = L->next;

List \*temp = p->next;

L->next = NULL;

while(p->next != NULL) //此处条件极其重要

{

p->next = L->next;

L->next = p;

p = temp;

temp = temp->next;

}

p->next = L->next;

L->next = p;

return OK;

}

int Merge\_list(List \*L1 , List \*L2)

{

if(!L1->next) //检查链表是否为空

{

cout << "请先创建联系人信息" << endl;

cout << "Wrong at Merge\_list()" << endl;

return ERROR;

}

List \*p = L1->next;

while(p->next != NULL)

{

p = p->next;

}

p->next = L2->next;

delete L2;

L2->next = NULL;

}

void menu() //菜单

{

cout << "1.创建" << endl; //finish

cout << "2.取值" << endl; //finish

cout << "3.查找" << endl; //finish

cout << "4.插入" << endl; //finish

cout << "5.删除" << endl; //finish

cout << "6.输出" << endl; //finish

cout << "7.逆序" << endl; //finish

cout << "8.合并" << endl; //finish

cout << "9.退出" << endl;

cout << "<---------------------------->" << endl;

}

int main()

{

char option = 0;

List \*L\_list[MAX] = {NULL};

int count = 0; //记录链表个数

int index = 0;

int first = 0; //用于合并操作

int second = 0; //用于合并操作

while(1)

{

menu();

cout << "请输入需要进行的操作：";

cin >> option;

fflush(stdin);

if(option < '1' || option > '9')

{

cout << "请输入正确的选项" << endl;

continue;

}

switch(option)

{

case '1':

if(count > 10)

{

cout << "没有足够空间" << endl;

cout << "请执行合并操作" << endl;

continue;

}

L\_list[count] = new List;

Get\_list(L\_list[count]);

cout << "<<--创建成功!-->>" <<endl;

count++;

// cout << count << endl;

break;

case '2':

cout << "请输入需要取值的链表（1-10）：";

cin >> index;

if(index <= 0)

{

cout << "输入不合法" << endl;

continue;

}

if(L\_list[index-1] == NULL) //该位置不存在链表

{

cout << "Wrong" << endl;

continue;

}

Out\_node(L\_list[index-1]);

break;

case '3':

cout << "请输入需要查找的链表（1-10）：";

cin >> index;

if(index <= 0)

{

cout << "输入不合法" << endl;

continue;

}

if(L\_list[index-1] == NULL) //该位置不存在链表

{

cout << "Wrong" << endl;

continue;

}

Find\_node(L\_list[index-1]);

break;

case '4':

cout << "请输入需要插入的链表（1-10）：";

cin >> index;

if(index <= 0)

{

cout << "输入不合法" << endl;

continue;

}

if(L\_list[index-1] == NULL) //该位置不存在链表

{

cout << "Wrong" << endl;

continue;

}

Insert\_node(L\_list[index-1]);

break;

case '5':

cout << "请输入需要删除的链表（1-10）：";

cin >> index;

if(index <= 0)

{

cout << "输入不合法" << endl;

continue;

}

if(L\_list[index-1] == NULL) //该位置不存在链表

{

cout << "Wrong" << endl;

continue;

}

Delete\_node(L\_list[index-1]);

break;

case '6':

cout << "请输入需要显示的链表（1-10）：";

cin >> index;

if(index <= 0)

{

cout << "输入不合法" << endl;

continue;

}

if(L\_list[index-1] == NULL) //该位置不存在链表

{

cout << "Wrong" << endl;

continue;

}

Show\_list(L\_list[index-1]);

break;

case '7':

cout << "请输入需要显示的链表（1-10）：";

cin >> index;

if(index <= 0)

{

cout << "输入不合法" << endl;

continue;

}

if(L\_list[index-1] == NULL) //该位置不存在链表

{

cout << "Wrong" << endl;

continue;

}

Opposite\_list(L\_list[index-1]);

break;

case '8':

cout << "请输入需要合并的两个链表位置（1-10）：" << endl;

cout << "第一个位置：";

cin >> first;

cout << "第二个位置：";

cin >> second;

if(first <= 0 || second <=0 || first == second)

{

cout << "输入不合法" << endl;

continue;

}

if(L\_list[first-1] == NULL || L\_list[second-1] == NULL) //该位置不存在链表

{

cout << "Wrong" << endl;

continue;

}

Merge\_list(L\_list[first-1] , L\_list[second-1]);

count--;

break;

case '9':

cout << "退出成功" << endl;

exit(1);

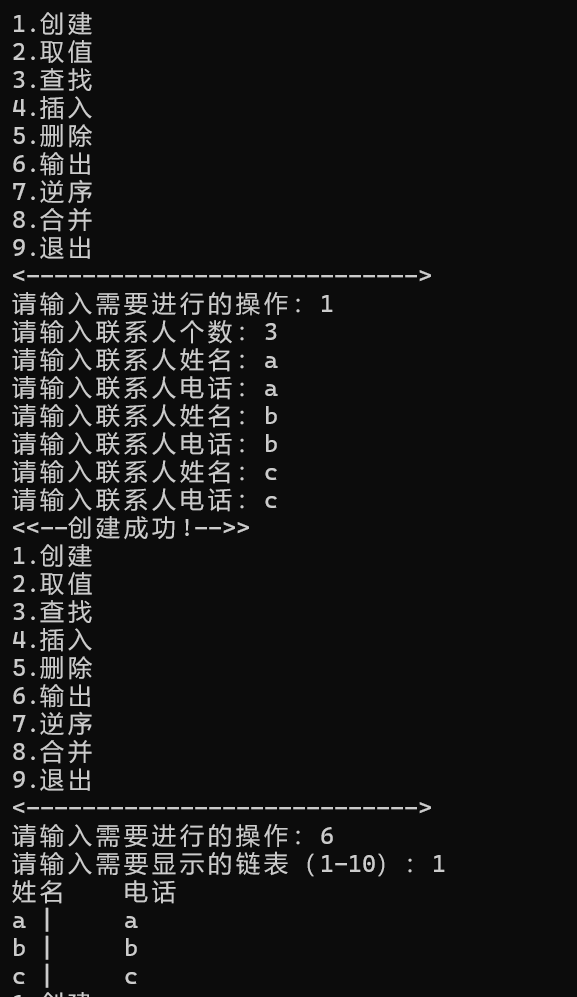
break;

}

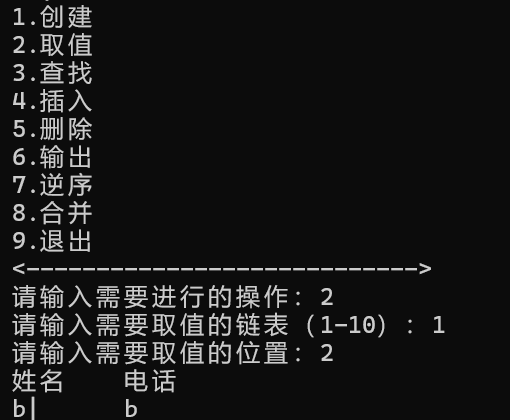
}

}

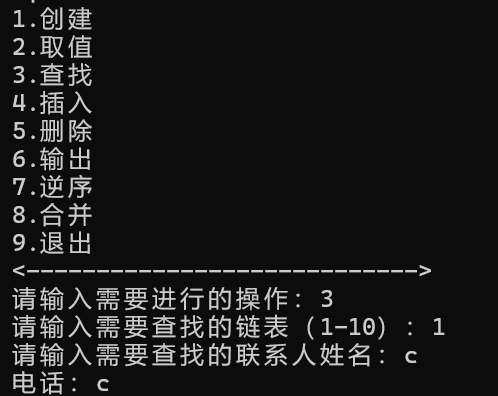
创建：



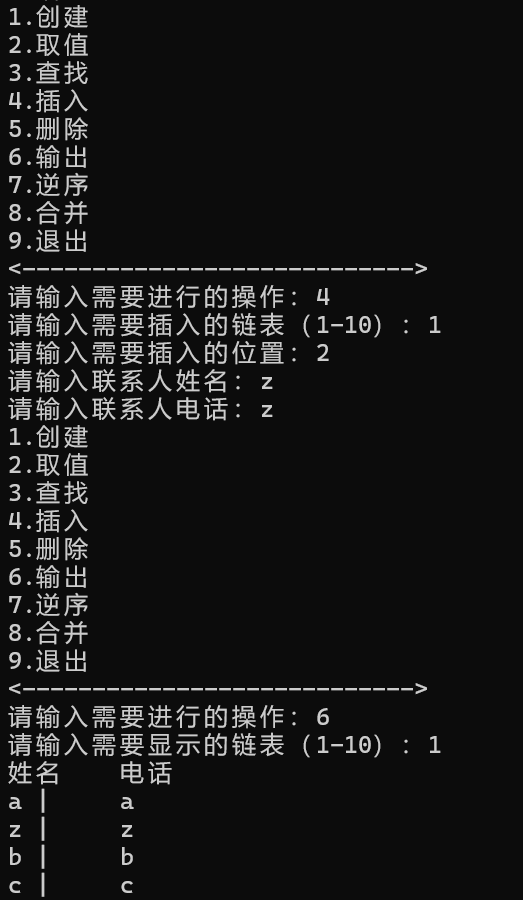
取值：



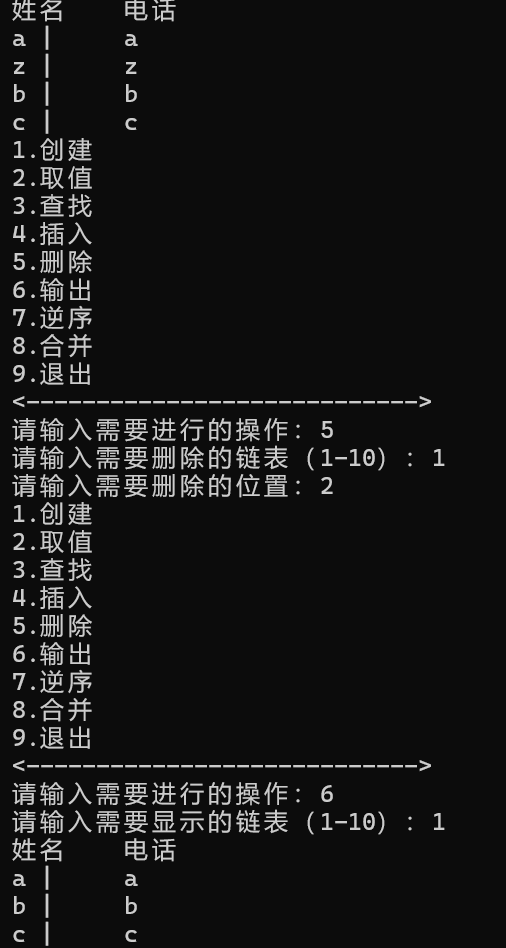
查找；



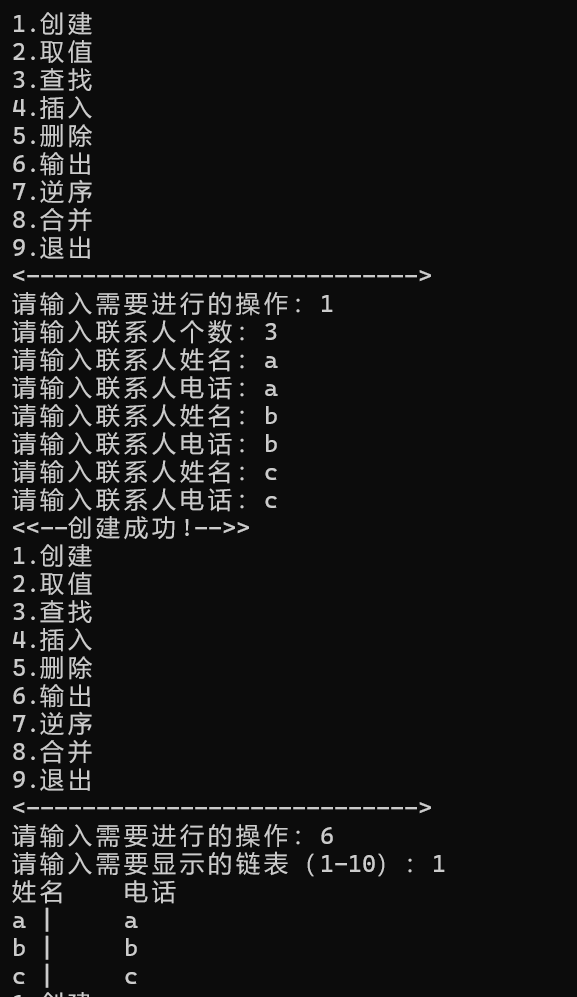
插入：



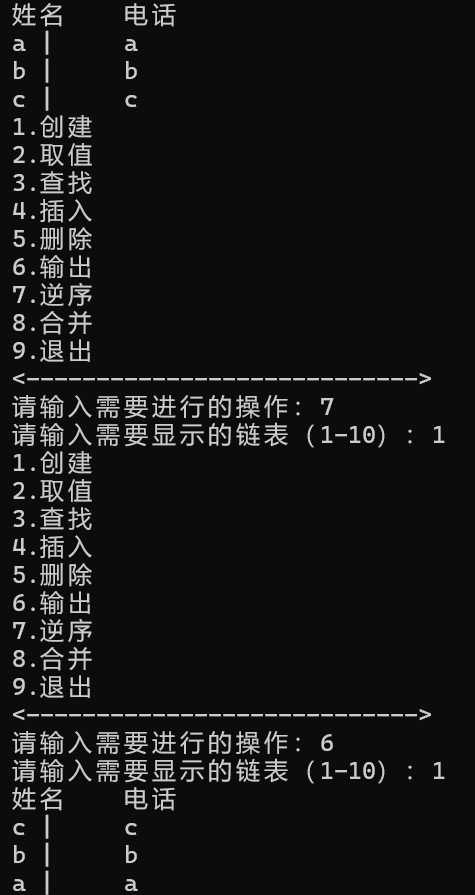
删除：



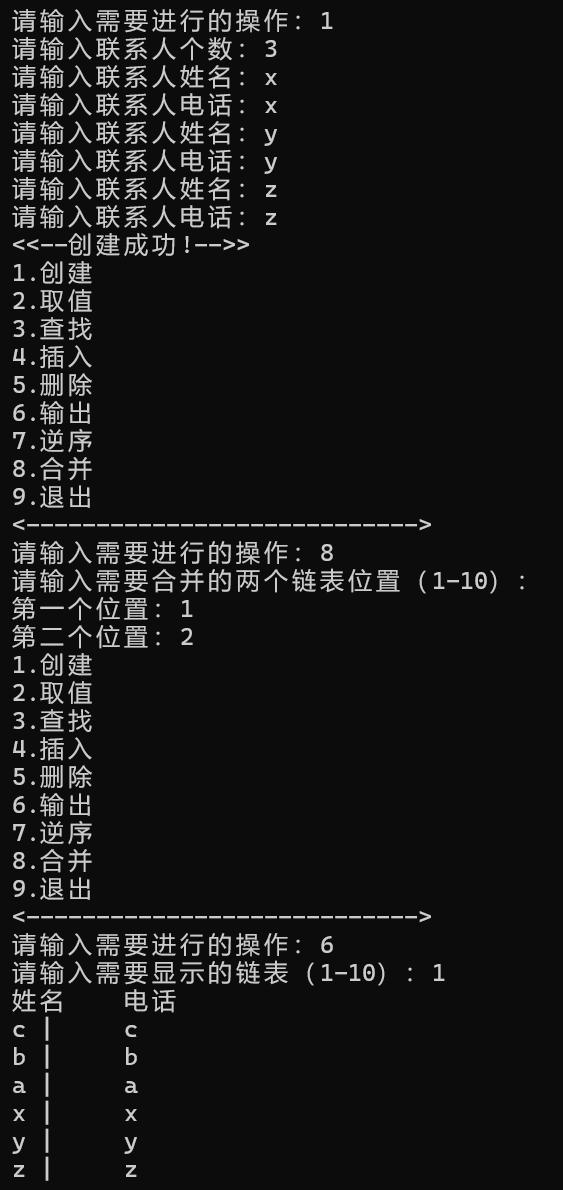
输出：



逆序：



合并：



退出：

