程序设计:

1. 请实现合并并反转两个**有序单向链表**的函数

* 链表节点存储的值为整数类型
* 输入参数为两个链表的头指针（或引用），链表中的节点已按值**从小到大**的顺序排列
* 合并两个链表中为一个链表，返回合并后的链表头指针（或引用），需要保证合并后链表中的节点按值**从大到小**的顺序排列
* 请自定义链表节点的结构

#pragma once

class ChainNode

{

public:

ChainNode(int);

ChainNode \*next;

int value;

private:

};

#include "ChainNode.h"

ChainNode::ChainNode(int value):

value(value),

next(nullptr)

{

}

#include "ChainNode.h"

#include <vector>

#include <iostream>

ChainNode \*reverse(ChainNode \*headNode) {

ChainNode \*preNode = nullptr;

ChainNode \*phead = nullptr;

ChainNode \*temp = nullptr;

phead = headNode;

while (phead != nullptr) {

temp = preNode;

preNode = phead;

phead = phead->next;//向下走一个

preNode->next = temp;//指向之前的小的

}

return preNode;

}

ChainNode \*mergeTwoChain(ChainNode \*first, ChainNode \*second) {

ChainNode \*newListHead = nullptr;

ChainNode \*newListRear = nullptr;

if (first->value <= second->value) {//选一个小的做头结点

newListHead = first;

newListRear = first;

first = first->next;

}

else

{

newListHead = second;

newListRear = second;

second = second->next;

}

while (first != nullptr && second != nullptr)

{

if (first->value <= second->value) {//向后走一个

newListRear->next = first;

newListRear = first;

first = first->next;

}

else

{

newListRear->next = second;

newListRear = second;

second = second->next;

}

}

if (first == nullptr) {

newListRear->next = second;//接上

}

if (second == nullptr) {

newListRear->next = first;

}

ChainNode \*result = reverse(newListHead);

return result;

}

int main() {

ChainNode \*a1 = new ChainNode(1);

ChainNode \*a2 = new ChainNode(6);

ChainNode \*a3 = new ChainNode(11);

a1->next = a2;

a2->next = a3;

ChainNode \*a4 = new ChainNode(2);

ChainNode \*a5 = new ChainNode(7);

ChainNode \*a6 = new ChainNode(10);

ChainNode \*a7 = new ChainNode(12);

a4->next = a5;

a5->next = a6;

a6->next = a7;

mergeTwoChain(a1, a4);

return 0;

}

（2）游戏中的每一轮，玩家将随机匹配到一个其他玩家的镜像进行对战，而且不会有玩家在同一轮被镜像两次。想知道对于给定的玩家人数，到底有多少种不同的匹配方式。

# 输入描述

一个整数N(N<=100)，表示给定的玩家数量。

# 输出描述

一个整数，表示可能的匹配种数

# **示例：**

## 输入

3

## 输出

2

## 说明

3名玩家A, B, C

可能有(A, B), (B, C), (C, A)和(A, C), (B, A), (C, B)两种匹配；

(A, A) (B, C) (C, B)为非法匹配，自己不能匹配自己;

(A, B), (B, C), (C, B)为非法匹配，B匹配了两次

#include <vector>

#include <iostream>

std::vector<int> usedNumList;

bool match(int curPlayerNum, int playerNum) {

if (curPlayerNum == playerNum) {

return false;//如果相等

}

if (std::find(usedNumList.begin(), usedNumList.end(), playerNum) != usedNumList.end()) {

return false;//如果后者用到过

}

std::cout << "(" << curPlayerNum << "," << playerNum << ")" << std::endl;

return true;

}

void tryToMatchPlayer(int curPlayerNum, int playerNum, int &matchCount) {

int matchedNum = 0;

for (int i = 1; i <= playerNum; i++) {

if (match(curPlayerNum, i)) {

if (curPlayerNum == playerNum) {

matchCount++;//该加一

}

matchedNum = i;

usedNumList.push\_back(i);

if (curPlayerNum < playerNum) {

tryToMatchPlayer(curPlayerNum + 1, playerNum, matchCount);

}

}

if (/\*i == playerNum &&\*/ matchedNum > 0) {

std::vector<int>::iterator iter = std::find(usedNumList.begin(), usedNumList.end(), matchedNum);

if (iter != usedNumList.end()) {//剔除掉

usedNumList.erase(iter);

matchedNum = 0;

}

}

}

}

void pickUpGame(int playerNum) {

usedNumList.clear();

int matchCount = 0;

tryToMatchPlayer(1, playerNum, matchCount);

}

int main() {

pickUpGame(4);

return 0;

}

系统设计:

用面向对象的思想设计一个文字交互类的迷宫游戏  
  
在迷宫中，玩家可以通过命令进行移动  
  
N 向北移动一格  
S 向南移动一格  
E 向东移动一格  
W 向西移动一格  
  
在移动过程中会遇到各种各样的可交互的物体，包括门，人物（NPC），或者怪物，同样玩家可以通过命令和这些所遇到的东西进行交互  
  
假设输入命令talk  
  
那么玩家就会尝试和当前物体进行交流，会和你说预设的台词，也可能不会理你，或者没有任何反应（比如门）  
  
假设输入命令action  
  
那么玩家就会尝试和当前物体进行交互，如果是门，就会尝试打开门，如果是怪物，就会开始攻击怪物，也可能有的物体什么都不会发生  
  
请设计和这些需求有关的类和接口（不需要具体的细节实现），并用这些类和其之间的关系，尝试描述如何来处理玩家的命令，处理玩家命令的函数如下  
  
void HandleCmd(string cmd)  
{  
  
}

迷宫有平面方格组成，每个方格为一个对象。抽象出格子基类，派生出不同类型的格子。

不同类型格子可以与玩家产生不同交互。

Enum CellTypeEnum{

Normal,

Door,

NPC,

Monster

}

Class CellDataBase{

Public:

CellTypeEnum type;

virtual void response();

}

Class CellDataNormal:CellDataBase{

//...

}

Class CellDataDoor:CellDataBase{

//...

}

地图类应该可以生成地图、加载地图、创建玩家。在玩家行走中，检测边界等。宽高等属性。

Class MapManager{

Public:

void CreateMap(int x, int y);

void loadMap(string & mapPath);

Player \* initPlayer();

Bool isOutOfBound(int x, int y);

Private:

Int columnNum;

Int rowNum;

}

玩家有静止、行走、说话、开门、攻击、倒下、胜利等行为。有处于地图中的坐标、生命、攻击力、朝向等属性。

Enum PlayerDirectionEnum{

North = 1,

South = 2;

West = 3;

East = 4;

}

Class Player{

Public:

Void idle();

Void move();

Void talk();

Void open();

Void attack();

Void defeat();

Void victory();

Private:

Int posX;

Int posY;

Int hp;

Int atk;

PlayerDirectionEnum dir;

}

玩家的行为由有限状态机管理。

Class State{//基类

Public:

Virtual void onEnter() = 0;

Virtual void execute() = 0;

Virtual void onExit() = 0;

}

Class IdleState{

void onEnter();

void execute();//播放玩家待机动作

void onExit();

}

Class MoveState{

void onEnter();

void execute();//播放玩家行走动作

void onExit();

}

Class Transition{//检查行为状态，比如行走时是否遇到阻碍、是否走向边界

Public:

Transition(State from, State to);

Bool check();

}

Class Command{//封装玩家指令

Public:

Command(std::string cmd);

Std::stringtring cmd;

Command(Command && cmd);

}

Class fsmManager{//状态机类，注册状态，注册状态间切换

Public:

Void Update(Command command);//执行一条指令

}

玩家操作应放入一个FIFO队列中，游戏逻辑循环从这个队列中取出指令，逐一处理。

Std::vector<Command> commandQueue;

输入指令时，将指令保存。如果输入和游戏逻辑循环，不在一个线程。要加个锁mutex

void HandleCmd(string cmd)  
{

Command cmd = Command(cmd);  
commandQueue.push\_back(cmd );  
}

游戏逻辑循环，从队列中拿出一条指令，驱动状态机进行状态检查

Void Update(dt){

If(commandQueue.size()>0){

Command cmd = commandQueue.erease(commandQueue.begin() );

fsmManager.Update(cmd);

}

}