TPP2022-HW1-40847031S 郭泰維

vec.h

```
#pragma once

// a 2d vector for math. not std::vector
class Vec {
public:
	int x, y;
	Vec();
	Vec(int x, int y);
	Vec(const Vec &rhs);
	Vec &operator=(const Vec &rhs);
	bool operator==(const Vec &rhs) const;
	bool operator!=(const Vec &rhs) const;
	bool operator<(const Vec &rhs) const;
	Vec operator+(const Vec &rhs) const;
	Vec operator-(const Vec &rhs) const;
	Vec &operator+=(const Vec &rhs);
	Vec &operator-=(const Vec &rhs);
};
```

`class Vec`

- 二維向量，方便在座標平面上做向量運算
- `int x`：水平座標
- `int y`：垂直座標

vec.cpp

```
#include "vec.h"

Vec::Vec(): x(0), y(0) {};
Vec::Vec(int x, int y): x(x), y(y) {}
Vec::Vec(const Vec &rhs): x(rhs.x), y(rhs.y) {};
Vec &Vec::operator=(const Vec &rhs) = default;
bool Vec::operator==(const Vec &rhs) const { return (x == rhs.x && y == rhs.y); }
bool Vec::operator!=(const Vec &rhs) const { return (x != rhs.x && y == rhs.y); }
bool Vec::operator<(const Vec &rhs) const { return x == rhs.x ? y < rhs.y : x < rhs.x; }
Vec Vec::operator+(const Vec &rhs) const { return Vec(x+rhs.x, y+rhs.y); }
Vec Vec::operator-(const Vec &rhs) const { return Vec(x-rhs.x, y-rhs.y); }
Vec &Vec::operator+=(const Vec &rhs) { return *this = *this + rhs; }
Vec &Vec::operator-=(const Vec &rhs) { return *this = *this - rhs; }
```

api.h

```
#pragma once

#include "vec.h"

// interface of cell that walkable
class IWalkableCell {
public:
	virtual bool isWalkable() const = 0;
};

// interface of maze
class IMaze {
public:
	virtual bool isWalkable(const Vec &loc) const = 0;
	virtual bool isInBound(const Vec &loc) const = 0;
	virtual int getHeight() const = 0;
	virtual int getWidth() const = 0;
	virtual const Vec &getRobotLocation() const = 0;
};
```

`class IWalkableCell`

- 迷宮格子的 interface，格子是組成迷宮的基本單位

`bool isWalkable() const`

- 確認這個格子是否為可以行走的

`class IMaze`

- 迷宮的 interface

`bool isWalkable(const Vec &loc) const`

- 確認是否可以行走到座標 `loc`

`bool isInBound(const Vec &loc) const`

- 確認座標 `loc` 是否出界

`int getHeight() const`

- 獲取迷宮高度

`int getWidth() const`

- 獲取迷宮寬度

`const Vec &getRobotLocation() const`

- 獲取機器人的初始位置

maze.h

```
#pragma once

#include <string>
#include <vector>
#include <map>
#include "api.h"

class Maze: public IMaze {
public:
	// initialization, templates should be implemented in the header file
	template<typename T>
	Maze(const T &src)
		: height_(src.size())
		, width_(height_ ? src[0].size() : 0)
		, cells_(height_, std::vector<const IWalkableCell*>(width_)) {
		for(int y = 0; y < height_; y++) {
			for(int x = 0; x < width_; x++) {
				switch(src[y][x]) {  // switch char at (x, y) to set cell
				case WALL: cells_[y][x] = &WallCell::getInstance(); break;
				case ROBOT: robot_loc_ = {x, y}; [[fallthrough]];
				default: cells_[y][x] = &FreeSpaceCell::getInstance(); break;
				}
			}
		}
	}
	virtual bool isWalkable(const Vec &loc) const override;
	virtual bool isInBound(const Vec &loc) const override;
	virtual int getHeight() const override;
	virtual int getWidth() const override;
	virtual const Vec &getRobotLocation() const override;
private:
	int height_, width_;
	Vec robot_loc_;
	std::vector<std::vector<const IWalkableCell*>> cells_;  // 2d std::vector of pointer to IWal
	static constexpr char WALL = '#', ROBOT = 'O';
	class FreeSpaceCell: public IWalkableCell {  // singleton free space cell
	public:
		virtual bool isWalkable() const override;
		static const FreeSpaceCell &getInstance();
	};
	class WallCell: public IWalkableCell {  // singleton wall cell
	public:
		virtual bool isWalkable() const override;
		static const WallCell &getInstance();
	};
};
```

`Maze`

- 繼承 `IMaze` ，使用二維 `std::vector` 儲存迷宮格子

`template<typename T> Maze(const T &src)`

- `Maze` 的建構子
- `const T &src`：原始資料， 可以接受傳入 `std::vector<std::vector<char>>` 或 `std::vector<std::string>`

`int height_`

- 迷宮的高度

`int width_`

- 迷宮的寬度

`std::vector<std::vector<const IWalkableCell*>> cells_`

- 迷宮主體，二維的迷宮格子

`static constexpr char WALL = '#', ROBOT = 'O'`

- 牆壁及機器人代表的字元

`class FreeSpaceCell`

- 空白的格子，使用單例模式

`const FreeSpaceCell &getInstance()`

- 回傳空白格子的實例

`class WallCell`

- 牆壁的格子，使用單例模

`const WallCell &getInstance()`

- 回傳牆壁格子的實例

Maze.cpp

```
#pragma once

#include "maze.h"

bool Maze::isWalkable(const Vec &loc) const {
	return cells_[loc.y][loc.x]->isWalkable();
}
bool Maze::isInBound(const Vec &loc) const {
	return loc.x >= 0 && loc.y >= 0
			&& loc.x < width_ && loc.y < height_;
}
int Maze::getHeight() const {
	return height_;
}
int Maze::getWidth() const {
	return width_;
}
const Vec &Maze::getRobotLocation() const {
	return robot_loc_;
}
bool Maze::FreeSpaceCell::isWalkable() const {
	return true;
}
const Maze::FreeSpaceCell &Maze::FreeSpaceCell::getInstance() {
	static const FreeSpaceCell instance;
	return instance;
}
bool Maze::WallCell::isWalkable() const {
	return false;
}
const Maze::WallCell &Maze::WallCell::getInstance() {
	static const WallCell instance;
	return instance;
}
```

Robot.h

```
#pragma once

#include <vector>
#include <map>
#include "api.h"

class Robot {
public:
	enum class EDirection {UP, RIGHT, DOWN, LEFT};
	Robot();
	const Vec &getLocation() const;
	const EDirection &getDirection() const;
	void setLocation(const Vec &loc);
	void setDirection(const EDirection &dir);
	void loadMaze(const IMaze *maze);
	bool boundFront() const;
	bool wallFront() const;
	void moveForward();
	void turnRight();
	void turnLeft();
private:
	Vec loc_;
	EDirection dir_;
	const IMaze *maze_;
	const Vec &frontVec() const;  // the front vector of robot
};
```

`class Robot`

- 機器人，可以移動以及轉彎

`enum class EDirection {UP, RIGHT, DOWN, LEFT}`

- 機器人的面向

`Robot()`

- 機器人的建構子

`const Vec &getLocation() const`

- 獲得機器人的位置

`const EDirection &getDirection() const`

- 獲得機器人的面向

`void setLocation(const Vec &loc)`

- 設置機器人的位置

`void setDirection(const EDirection &dir)`

- 設置機器人的面向

`void loadMaze(const IMaze *maze)`

- 載入迷宮

`bool boundFront() const`

- 確認前方是否為迷宮邊界

`bool wallFront() const`

- 確認前方是否有牆壁

`void moveForward()`

- 使機器人向前移動

`void turnRight()`

- 使機器人向右轉

`void turnLeft()`

- 使機器人向左轉

`Vec loc_`

- 機器人的位置

`EDirection dir_`

- 機器人的面向

`const IMaze *maze_`

- 已經載入的迷宮

`const Vec &frontVec() const`

- 回傳機器人面向的單位向量

Robot.cpp

```
#include "robot.h"

Robot::Robot(): loc_(), dir_(EDirection::UP), maze_(nullptr) {};
const Vec &Robot::getLocation() const { return loc_; }
const Robot::EDirection &Robot::getDirection() const { return dir_; }
void Robot::setLocation(const Vec &loc) { loc_ = loc; }
void Robot::setDirection(const EDirection &dir) { dir_ = dir; }
void Robot::loadMaze(const IMaze *maze) {
	maze_ = maze;
	loc_ = maze_->getRobotLocation();
}
bool Robot::boundFront() const {
	return !maze_->isInBound(loc_+frontVec());
}
bool Robot::wallFront() const {
	return !maze_->isWalkable(loc_+frontVec());
}
void Robot::moveForward() {
	loc_ += frontVec();
}
void Robot::turnRight() {
	dir_ = static_cast<EDirection>((static_cast<int>(dir_)+1)%4);
}
void Robot::turnLeft() {
	dir_ = static_cast<EDirection>((static_cast<int>(dir_)+3)%4);
}
const Vec &Robot::frontVec() const {
	static const Vec UP(0, -1), RIGHT(1, 0), DOWN(0, 1), LEFT(-1, 0);
	switch(dir_) {
	case EDirection::UP: return UP; break;
	case EDirection::RIGHT: return RIGHT; break;
	case EDirection::DOWN: return DOWN; break;
	case EDirection::LEFT: return LEFT; break;
	}
	return UP;
}
```

main.cpp

```
#include <iostream>
#include "robot.h"
#include "maze.h"

int main() {
	int w, h;
	long long n;
	std::cin >> w >> h >> n; std::cin.ignore();

	std::vector<std::string> lines(h);
	for (auto &line: lines) std::getline(std::cin, line);

	Maze maze(lines);
	Robot robot;
	robot.loadMaze(&maze);

	auto &&res = [&n, &robot]() mutable {
		int step = 0;
		std::map<std::pair<Vec, Robot::EDirection>, int> history;  // get step by status fro
		std::vector<std::pair<Vec, Robot::EDirection>> path;  // get status by step
		while(step < n) {
			// turn right if can not walk forward
			while(robot.boundFront() || robot.wallFront()) robot.turnRight();

			// make status
			auto &&status = std::make_pair(robot.getLocation(), robot.getDirection());

			// found if status has appeared in history
			if(auto &&found = history.find(status); found != history.end()) {
				// reduce loop step
				int pre = found->second;
				n = pre + (n-pre) % (step-pre);
				return path[n].first;
			}
			history.emplace(status, step);
			path.emplace_back(status);
			robot.moveForward();
			step++;
		}
		return robot.getLocation();
	}();

	std::cout << res.x << ' ' << res.y << std::endl;
}
```

程式解釋

`int w, h`

- 迷宮的寬高

`lnog long n`

- 機器人可以移動的最大步數

`std::vector<std::string> lines(h)`

- 迷宮的輸入

`Maze maze(lines)`

- 迷宮

`Robot robot`

- 機器人

`auto &&res`

- 機器人在迷宮移動完的最後位置

`int step = 0`

- 機器人已經移動的步數

`std::map<std::pair<Vec, Robot::EDirection>, int> history`

- 歷史紀錄，紀錄機器人在之前在這個位置及方向時的步數

`std::vector<std::pair<Vec, Robot::EDirection>> path`

- 路徑，紀錄機器人在每一步的位置及方向

`auto &&status`

- 機器人的狀態，含有位置及方向

`auto &&found`

- 搜尋歷史紀錄的結果，如果找不到，會得到 history.end()

過程

1. 輸入迷宮的寬高以及最大步數，並讀取迷宮資料
2. 建立 Maze 及 Robot
3. 當機器人當前步數小於最大步數時:
    i. 如果不能往前走，則右轉
    ii. 檢查當前狀態是否出現在歷史紀錄
        a. 如果是，則將循環的步數(當前步數-歷史步數)扣掉，計算出最後的位置
        b. 否則，記錄當前狀態並向前走直到達到最大步數
4. 輸出機器人最後的位置