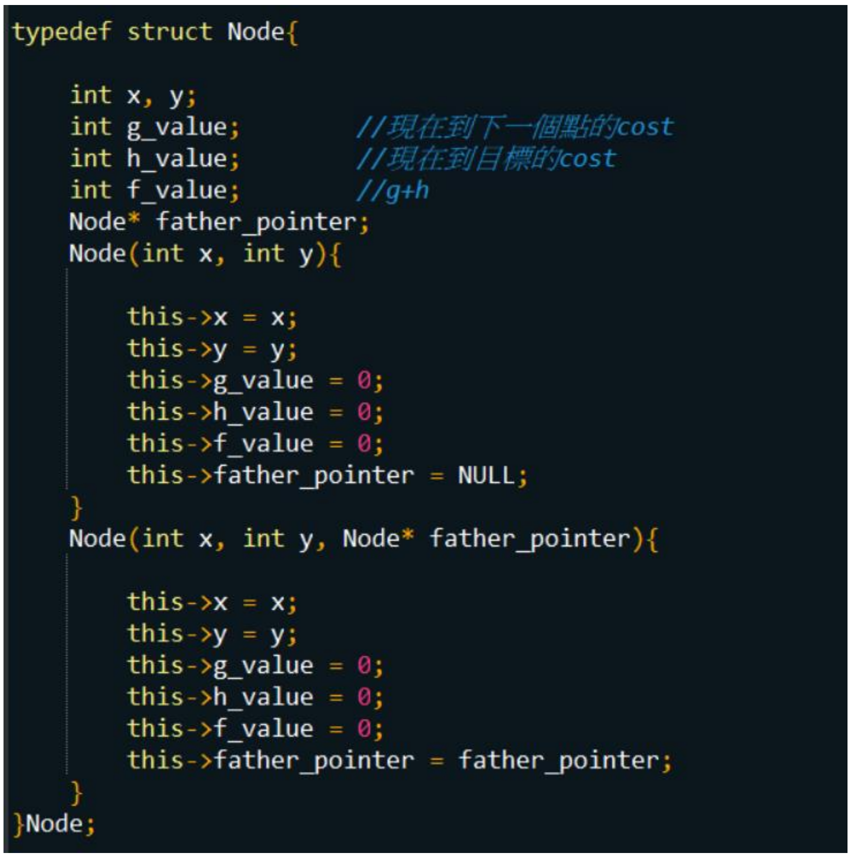
初始化設定

1. **走訪的節點結構**



```
typedef struct Node{

    int x, y;
    int g_value;        //現在到下一個點的cost
    int h_value;        //現在到目標的cost
    int f_value;        //g+h
    Node* father_pointer;
    Node(int x, int y){

        this->x = x;
        this->y = y;
        this->g_value = 0;
        this->h_value = 0;
        this->f_value = 0;
        this->father_pointer = NULL;
    }
    Node(int x, int y, Node* father_pointer){

        this->x = x;
        this->y = y;
        this->g_value = 0;
        this->h_value = 0;
        this->f_value = 0;
        this->father_pointer = father_pointer;
    }
}Node;
```

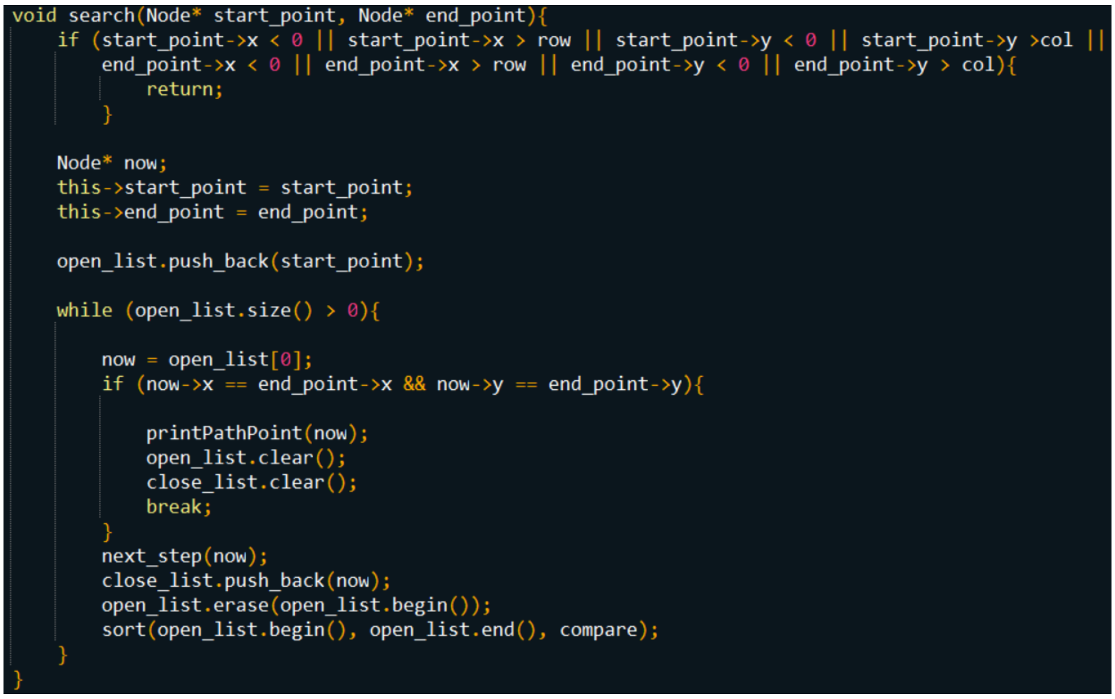
x 和 y 表示該點座標位置

g_value、h_value 和 f_value 為 A*演算法走訪的依據

father_pointer 為距離最短的父節點指標

2. Astar class 中

(1) search function



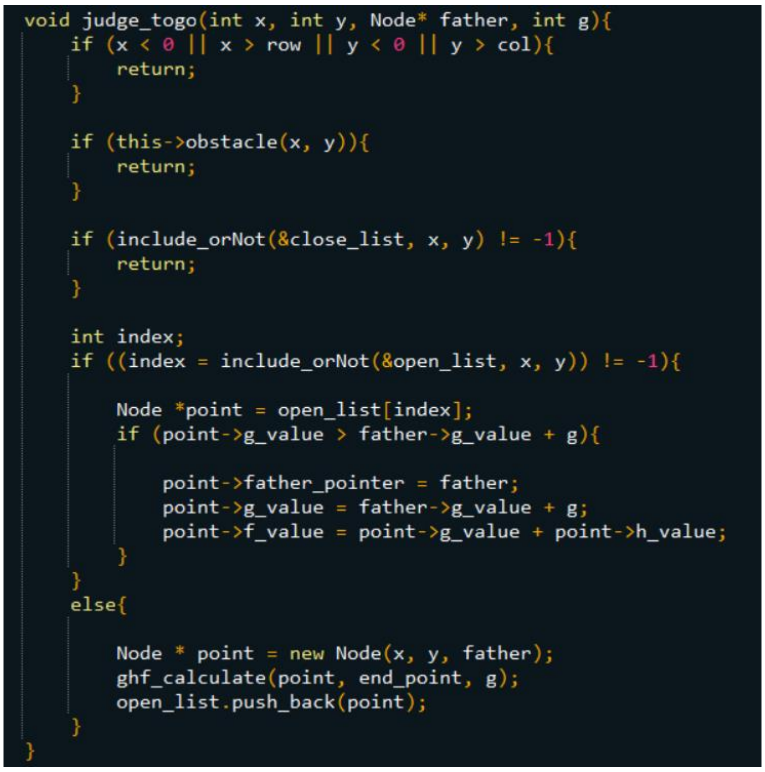
```
void search(Node* start_point, Node* end_point){
    if (start_point->x < 0 || start_point->x > row || start_point->y < 0 || start_point->y >col ||
        end_point->x < 0 || end_point->x > row || end_point->y < 0 || end_point->y > col){
            return;
        }

    Node* now;
    this->start_point = start_point;
    this->end_point = end_point;

    open_list.push_back(start_point);

    while (open_list.size() > 0){

        now = open_list[0];
        if (now->x == end_point->x && now->y == end_point->y){

            printPathPoint(now);
            open_list.clear();
            close_list.clear();
            break;
        }
        next_step(now);
        close_list.push_back(now);
        open_list.erase(open_list.begin());
        sort(open_list.begin(), open_list.end(), compare);
    }
}
```

此 function 為主要的走訪函式

第一個判斷目的為不超過邊界，接下來將起始點放入 open_list 中，
即開始走訪，

若走到 end_point(終點)則印出路線並將 list 清空，跳出迴圈

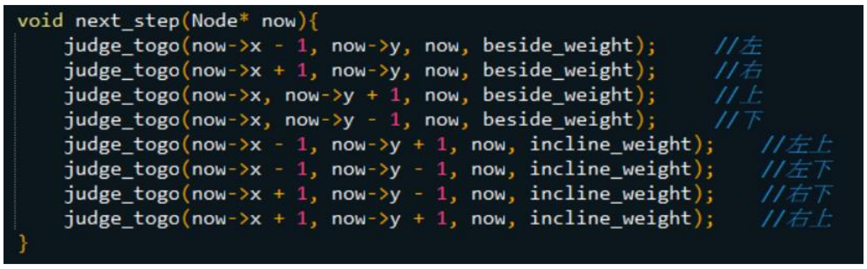
(2) judge_togo function



```
void judge_togo(int x, int y, Node* father, int g){
    if (x < 0 || x > row || y < 0 || y > col){
        return;
    }

    if (this->obstacle(x, y)){
        return;
    }

    if (include_orNot(&close_list, x, y) != -1){
        return;
    }

    int index;
    if ((index = include_orNot(&open_list, x, y)) != -1){

        Node *point = open_list[index];
        if (point->g_value > father->g_value + g){

            point->father_pointer = father;
            point->g_value = father->g_value + g;
            point->f_value = point->g_value + point->h_value;
        }
    }
    else{

        Node * point = new Node(x, y, father);
        ghf_calculate(point, end_point, g);
        open_list.push_back(point);
    }
}
```

此 function 的目的為判斷此點能不能被走訪，若可以則加入到 open_list 中

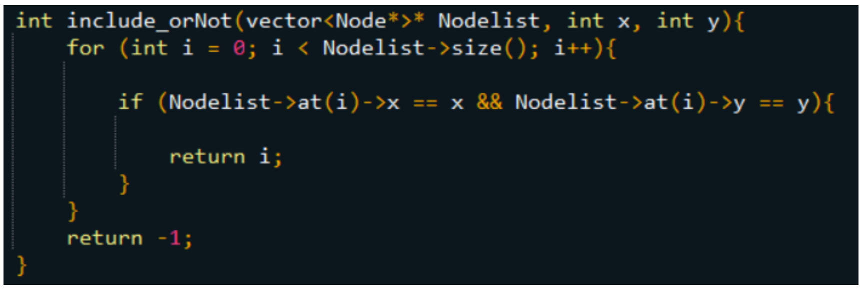
(3) next_step function



```
void next_step(Node* now){
    judge_togo(now->x - 1, now->y, now, beside_weight);     //左
    judge_togo(now->x + 1, now->y, now, beside_weight);     //右
    judge_togo(now->x, now->y + 1, now, beside_weight);     //上
    judge_togo(now->x, now->y - 1, now, beside_weight);     //下
    judge_togo(now->x - 1, now->y + 1, now, incline_weight);    //左上
    judge_togo(now->x - 1, now->y - 1, now, incline_weight);    //左下
    judge_togo(now->x + 1, now->y - 1, now, incline_weight);    //右下
    judge_togo(now->x + 1, now->y + 1, now, incline_weight);    //右上
}
```

由現在所在的點出發，往八個方向各走一格，並測試是否為可以被走訪的點

(4) include_orNot function

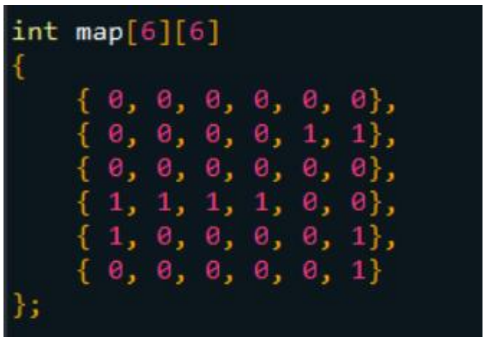


```
int include_orNot(vector<Node*>* Nodelist, int x, int y){
    for (int i = 0; i < Nodelist->size(); i++){

        if (Nodelist->at(i)->x == x && Nodelist->at(i)->y == y){

            return i;
        }
    }
    return -1;
}
```

判斷是否已經在 open_list 中，若有就回傳它的 index

程式執行

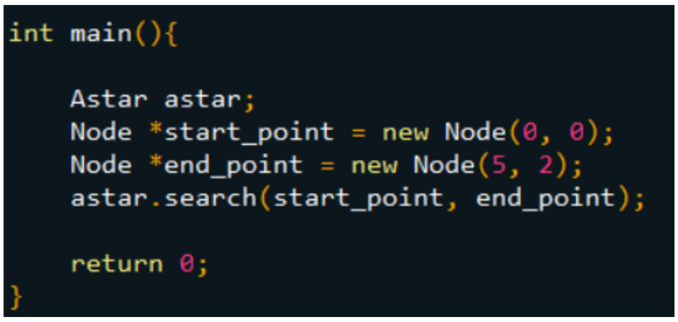
1. 地圖設定



```
int map[6][6]
{
    { 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 1, 1},
    { 0, 0, 0, 0, 0, 0},
    { 1, 1, 1, 1, 0, 0},
    { 1, 0, 0, 0, 0, 1},
    { 0, 0, 0, 0, 0, 1}
};
```

0 表示可以走的地方，1 表示障礙物

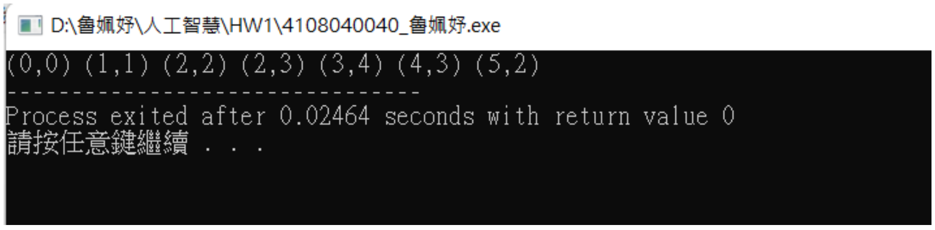
2. Main function 和起始點設定



```
int main(){

    Astar astar;
    Node *start_point = new Node(0, 0);
    Node *end_point = new Node(5, 2);
    astar.search(start_point, end_point);

    return 0;
}
```

start_point 為起始點，end_point 為目標點(終點)

執行結果



D:\魯姵妤\人工智慧\HW1\4108040040_魯姵妤.exe

```
(0,0) (1,1) (2,2) (2,3) (3,4) (4,3) (5,2)
--------------------------------
Process exited after 0.02464 seconds with return value 0
請按任意鍵繼續 . . .
```