链表逆置

```
#include<iostream>
#include<fstream>
#include<string>
#include<vector>
#include<unordered_map>
using namespace std;
struct node
{
    int data;
    node *next;
};
node* reverse(node *head)
{
    node *pre = NULL;
    node *next = head;
    node *p = head;
    while(p!=NULL)
    {
        next = p->next;
        p->next = pre;
        pre = p;
        p = next;
    }
    return pre;
}
int main()
{
    node *a = new node;
    node *b = new node;
    node *c = new node;
    a->data = 1;

    a->next = b;
    b->data = 2;
    b->next = c;
    c->data = 3;
    c->next = NULL;
    node *head = reverse(a);
    while(head!=NULL)
    {
        cout<<head->data<<endl;;
        head =head->next;
    }
    return 0;
}
```

文件读写

```
//不能同时用 ifstream 和 ofstream 打开同一个文件。
#include <iostream>
#include <fstream>
#include <string>
#include<vector>>
//不能同时ifstream和ofstream

/*
mapper.txt
Hold fast to dreams
For if dreams die
Life is a broken-winged bird

*/
using namespace std;
int main() {


    ifstream file("mapper.txt");  // 使用 / 替代
    if (!file)
    {
        cout << "打开失败";
    }

    vector<string> ss;
    string line;
    while (getline(file, line))
    {
        string aaa = "";
        for (int i = 0;i < line.size();i++)
        {
            if (line[i] != ' ')
            {
                aaa += line[i];
            }
            else if (line[i] = ' ')
            {
                aaa += '\0';
                ss.push_back(aaa);
                aaa = "";
            }
        }

        if (aaa != "")
        {
            aaa += '\0';
            ss.push_back(aaa);
        }
    }
    for (auto i : ss)
    {
        cout << i << endl;
    }
```

```
    file.close();  // 关闭文件


    ofstream file1("wawa.txt");
    if (file1)
    {
        cout << "打开成功";
    }
    for (int i = 0;i < ss.size();i++)
    {
        file1 << ss[i];
    }
    file1.close();



    return 0;

}
```

模板类实现栈

```
#include<vector>
#include<iostream>
using namespace std;
template<typename T>
class Mystack {
private:
    vector<T> elems;
public:
    void push(T&& e);
    void pop(T& e);
    int size() const;
    bool empty() const;
};

template<typename T>
void Mystack<T>::push(T&& a)                    //注意模板类函数的写法
（Mystack<T>::push）
{
    elems.push_back(a);
}

template<typename T>
void Mystack<T>::pop(T& a)
{
    a = elems.back();                           //注意：取出vector最后一个元素
    elems.pop_back();
}

template<typename T>
int Mystack<T>::size() const
{
    return elems.size();
```

```
}

template<typename T>
bool Mystack<T>::empty() const
{
    if (elems.size() == 0)
    {
        return true;
    }
    else
    {
        return false;
    }

}

int main()
{
    Mystack<int> s;
    s.push(10);
    s.push(20);
    s.push(30);
    cout << s.size() << endl;
    while (!s.empty())
    {
        int e;
        s.pop(e);
        cout << e << endl;
    }
    return 0;
}
```

多态对象指针

```
#include<iostream>
using namespace std;
class USB {
public:
    virtual void read() = 0;
    virtual ~USB() {}
};

class Computer
{
private:
    USB* pdevice;                    //对象指针
public:
    void setUSB(USB* p)              //初始化对象指针
    {
        pdevice = p;

    };

    virtual void read()            //对象指针调用对象
```

```
    {
        pdevice->read();
    }


};
class Disk :public USB
{
    void read()
    {
        cout << "从硬盘中读取数据" << endl;
    }
};
class Camera :public USB
{
    void read()
    {
        cout << "从相机中读取数据" << endl;
    }
};
class Scanner :public USB
{
    void read()
    {
        cout << "从扫描仪中读取数据" << endl;
    }
};
int main() {
    Computer computer;
    //从Disk读取
    USB* usb = new Disk();
    computer.setUSB(usb);
    computer.read();
    delete usb;
    //从Camera读取
    usb = new Camera();
    computer.setUSB(usb);
    computer.read();
    delete usb;
    //从Scanner读取
    usb = new Scanner();
    computer.setUSB(usb);
    computer.read();
    delete usb;
    return 0;
}
```

字符串提取数字

```
#include<iostream>
#include<vector>
#include<cmath>
using namespace std;
double to_double(string ss)        //将字符串转化为数字
{
    int flag = 0;
    double sum = 0;
    int nn = 1;
    for (int i = 0;ss[i] != '\0';i++)
    {
        if (ss[i] == '.')
        {
            flag = 1;
        }
        else if (flag == 0)
        {
            sum = sum * 10 + int(ss[i] - '0');     //注意：int('1'),返回的是'1'的
askcll码，
        }
        else if (flag == 1)
        {
            sum = sum + pow(0.1, nn) * int(ss[i] - '0');
            nn = nn + 1;
        }
    }
    return sum;

}
int main()
{
    vector<string> number;
    double count = 0;
    string ss;
    getline(cin, ss);
    string aa = "";
    for (int i = 0;i < ss.size();i++)
    {

        if ((ss[i] >= '0' && ss[i] <= '9') || ss[i] == '.')
        {
            aa += ss[i];
        }
        else if (aa.size() != 0)
        {
            number.push_back(aa);
            aa = "";
        }
    }
    if (aa.size() != 0)
    {
        number.push_back(aa);
    }
```

```
    for (auto x : number)
    {
        cout << x << "          " << to_double(x) << endl;
        count += to_double(x);
    }
    cout << count << endl;
    return 0;
}
```

重载运算符

```
#include<iostream>
#include<vector>
#include<string>
#include<unordered_map>
using namespace std;
class Point
{
protected:
    int x;
    int y;
public:
    Point() {};
    Point(int x, int y) :x(x), y(y) {};
    Point(const Point& a)                        //复制构造函数 1、const  2、类名  3、对象
    {
        cout << "复制构造函数" << endl;
        this->x = a.x;
        this->y = a.y;
    }
    friend istream& operator>>(istream& input, Point& p)  //重载输入输出  1、友元 2、istream &input  3、Point &p    4、input>>p.x>>p.y  5、return input
    {
        cout << "输入x和y:";
        input >> p.x >> p.y;
        return input;
    }
    friend ostream& operator<<(ostream& output, Point& p)
    {
        output << '(' << p.x << ',' << p.y << ')' << endl;
        return output;
    }

    /*Point operator +(Point &a)                    //重载成员函数，一个参数是调用的对象，另一个参数是传进来的
    {
        return Point(a.x+this->x,a.y+this->y);
    }*/
    friend Point operator+(Point a, Point b);
    void display()
    {
        cout << '(' << this->x << ',' << this->y << ')' << endl;
    }
    /*
```

```
    Point operator=()                                   //重载等号运算符：1、必须成员函数
    {
        "重载等号运算符只能用成员函数"
        return Point(this->x,this->y);
    }
    */


};
Point operator+(Point a, Point b)                 //Point Point::operator +(Point &a,Point &b)注意：友元函数不是类的成员，不需要::
{                                                 //另外：友元函数的参数不能是引用类型！！！


    cout << "重载友元函数+" << endl;
    return Point(a.x + b.x, a.y + b.y);
}
int main()
{
    Point a(1, 2);
    Point b(3, 4);
    Point c;
    cin >> c;
    cout << c;
    //Point c = a+b;
    c.display();
    return 0;
}
```

学生成绩排序

```
#include<iostream>
#include<vector>
using namespace std;
class stu
{
public:
    int score;
    int id;
    stu() {}
    stu(int id, int score) :id(id), score(score) {}
    void display()
    {
        cout << "id:" << id << "    " << "score:" << score << endl;
    }
    stu(const stu& a) :id(a.id), score(a.score) {}                  //注意：构造函数的写法：1、const2、&

};
//注意：用引用，否则排序不成功
void sort(vector<stu>& aaa)                                         //注意：使用引用，否则排序不成功
{
```

```
    for (int i = 0;i < aaa.size();i++)
    {
        for (int j = 0;j < i;j++)
        {
            if (aaa[i].score > aaa[j].score)
            {
                stu temp;
                temp = aaa[i];
                aaa[i] = aaa[j];
                aaa[j] = temp;

            }
        }
    }
}
//
//void sort(vector<stu>& aaa)  // 用引用，避免拷贝 vector
//{
//  int n = aaa.size();
//  for (int i = 0; i < n - 1; i++)
//  {
//      for (int j = 0; j < n - i - 1; j++)  // 修正范围
//      {
//          if (aaa[j].score < aaa[j + 1].score) // 交换位置
//          {
//              swap(aaa[j], aaa[j + 1]);
//          }
//      }
//  }
//}

int main()
{
    int num;
    cin >> num;
    vector<stu> student;
    for (int i = 0;i < num;i++)
    {
        int id;
        int score;
        cin >> id >> score;
        student.push_back(stu(id, score));                              //注意：新建
vector，内部为存储对象
    }
    sort(student);
    for (auto i : student)
    {
        i.display();
    }
    return 0;
}
```

学生类

```
#include<iostream>
#include<vector>
#include<string>
#include<cstring>                                   //注意：strcpy包含在cstring里面
#define _CRT_SECURE_NO_WARNINGS

using namespace std;
class stu;
class tea
{
public:
    int id;
    void change(stu& a);                            //不能直接在这里实现change()，因为stu
类还没有定义出来。
    tea(int id) :id(id) {}
};


class stu
{
protected:                                          //为了让派生类能够访问，修改权限为
protected

    char* name;
    int score;

public:
    static int count;
    friend void tea::change(stu&);
    stu()
    {
        this->name = new char[100];                     //无参构造函数，要分派数组的空间
        count++;
    }
    stu(char* name, int score) :score(score)            //构造函数
    {
        this->name = new char[100];
        strcpy(this->name, name);
        count++;
    }
    stu(const stu& a) :score(a.score) {                  //复制构造函数 1、const  2、
&   3指针要分派新的空间

        this->name = new char[100];
        strcpy(this->name, a.name);
        count++;

    }
    void display()
    {
        cout << this->name << ' ' << this->score << endl;
    }
```

```
    static void showcount()                                    //静态成员函数，用于访问静态成
员变量
    {
        cout << count;
    }
    ~stu()
    {
        delete[]name;                                          //析构函数
        count--;
    }

};
int stu::count = 0;                                            //类外初始化静态成员函数

void tea::change(stu& a)
{
    int xxx;
    cin >> xxx;

    a.score = xxx;
}

class goodstu : public stu
{
    int id;
public:
    goodstu(int id, char* name, int score) :id(id), stu(name, score)
    {
        this->name = new char[100];
        strcpy(this->name, name);
        count++;
    }
    ~goodstu() { count--; }

};
int main()
{
    cout << "输入学生数量: ";
    int num;
    cin >> num;
    vector<stu> aaa;   //对象数组
    for (int i = 0;i < num;i++)
    {
        char name[10];   //字符数组
        int score;
        cin >> name >> score;
        aaa.push_back(stu(name, score));
    }
    for (auto i : aaa)
    {
        i.display();
    }
    cout << "showcount:";
    stu::showcount();
    cout << endl << "input student id to change:";
```

```
    int id;
    cin >> id;
    tea tt(id);
    tt.change(aaa[id]);
    for (auto i : aaa)
    {
        i.display();
    }

}
```

大数相加

```
#include <iostream>
#include <string>
#include<algorithm>
using namespace std;
class HugeInteger
{
public:
    string ss;
    HugeInteger(int aaa)
    {
        ss = to_string(aaa);
    }
    HugeInteger() { this->ss = ""; };
    HugeInteger(string aaa) :ss(aaa) {};
    friend ostream& operator<<(ostream& output, HugeInteger aaa)
    {
        output << aaa.ss;
        return output;
    }
    HugeInteger operator +(HugeInteger aa)
    {
        string ret;
        string xx = aa.ss;
        string yy = this->ss;
        reverse(xx.begin(), xx.end());
        reverse(yy.begin(), yy.end());
        int maxit = max(xx.size(), yy.size());
        if (xx.size() < maxit)
        {
            for (int i = xx.size();i < yy.size();i++)
            {
                xx += '0';
            }xx += '\0';
        }
        else {
            for (int i = yy.size();i < xx.size();i++)
            {
                yy += '0';
            }yy += '\0';
```

```
        }
        int nextlevel = 0;
        int thislevel = 0;
        for (int i = 0;i < maxit;i++)
        {
            thislevel = int(xx[i] - '0') + int(yy[i] - '0') + nextlevel;
            nextlevel = 0;
            nextlevel = thislevel / 10;

            thislevel = thislevel % 10;
            ret += to_string(thislevel);
        }
        if (nextlevel != 0)ret += to_string(nextlevel);
        ret += '\0';

        reverse(ret.begin(), ret.end());
        return (ret);
    }
    HugeInteger operator +(string aa)
    {
        string ret;
        string xx = aa;
        string yy = this->ss;
        reverse(xx.begin(), xx.end());
        reverse(yy.begin(), yy.end());
        int maxit = max(xx.size(), yy.size());
        if (xx.size() < maxit)
        {
            for (int i = xx.size();i < yy.size();i++)
            {
                xx += '0';
            }xx += '\0';
        }
        else {
            for (int i = yy.size();i < xx.size();i++)
            {
                yy += '0';
            }yy += '\0';
        }
        int nextlevel = 0;
        int thislevel = 0;
        for (int i = 0;i < maxit;i++)
        {
            thislevel = int(xx[i] - '0') + int(yy[i] - '0') + nextlevel;
            nextlevel = 0;
            nextlevel = thislevel / 10;

            thislevel = thislevel % 10;
            ret += to_string(thislevel);
        }
        if (nextlevel != 0)ret += to_string(nextlevel);
        ret += '\0';

        reverse(ret.begin(), ret.end());
        return (ret);
```

```
    }

};



int main()
{
    HugeInteger n1(7654321);
    HugeInteger n2(7891234);
    HugeInteger n3("99999999999999999999999999999");
    HugeInteger n4("1");
    HugeInteger n5;
    cout << "n1 is " << n1 << "\nn2 is " << n2
        << "\nn3 is " << n3 << "\nn4 is " << n4
        << "\nn5 is " << n5 << "\n\n";
    n5 = n1 + n2;
    cout << n1 << " + " << n2 << " = " << n5 << "\n\n";
    cout << n3 << " + " << n4 << "\n= " << (n3 + n4) << "\n\n";
    n5 = n1 + 9;
    cout << n1 << " + " << 9 << " = " << n5 << "\n\n";
    n5 = n2 + "10000";
    cout << n2 << " + " << "10000" << " = " << n5 << endl;

    return 0;
}
```

回溯

n皇后

```
#include<iostream>
#include<vector>
#include<set>
#include<unordered_map>
using namespace std;
int count = 0;
/*
输入棋盘大小：6
输出所有的情况，即从第1行到第6行棋子对应的列坐标
1 3 5 0 2 4             第一种情况，棋子对应坐标为(0,1)(1,3)(2,5)(3,0)(4,2)(5,4)
2 5 1 4 0 3
3 0 4 1 5 2
4 2 0 5 3 1
*/
vector<vector<int>> result;             //保存坐标，一个vector<int>代表一种情况

void display(vector<vector<char>> a)
{
    cout << endl;
    for (auto aaa : a)
```

```
    {
        for (int i = 0;i < aaa.size();i++)
        {
            cout << aaa[i] << ' ';
        }cout << endl;
    }
}
void f(vector<vector<char>>& borad, unordered_map<int, int>column,
unordered_map<int, int>set45, unordered_map<int, int>set135, int num, int row,
vector<int> path)
{
    if (row == num)                                                    //递归中止情
况：行数
    {
        result.push_back(path);
    }
    for (int col = 0;col < num;col++)
    {
        if (column[col] == 0 && set45[col + row] == 0 && set135[col - row] == 0)
   //列，45，135均无冲突时
        {
            column[col] = 1;                                           //标记列冲突
            set45[col + row] = 1;
            set135[col - row] = 1;
            borad[row][col] = 'Q';
            path.push_back(col);
            f(borad, column, set45, set135, num, row + 1, path);       //每深入一
层递归行数+1
            column[col] = 0;
            set45[col + row] = 0;                                      //回溯
            set135[col - row] = 0;
            borad[row][col] = '.';
            path.pop_back();

        }
    }
    display(borad);
}
using namespace std;
int main()
{
    int num;
    cout << "input num of rows:" << endl;
    cin >> num;
    vector<int> path;                                                  //记录该路径的列坐标

    unordered_map<int, int> set45;                                     //记录目前所有的
45°对角线，同一个45°对角线上的元素横纵坐标和相等，只要hash[行坐标+列坐标] = 1即代表该对角线已
被使用
    unordered_map<int, int> set135;                                    //记录目前所有的
135°对角线，同一个135°对角线上的元素横纵坐标差相等，只要hash[行坐标-列坐标] = 1即代表该对角线
已被使用
    unordered_map<int, int> column;                                    //记录已被使用的列
    vector<vector<char>> borad(num, vector<char>(num, '.'));
    display(borad);
```

```
    f(borad, column, set45, set135, num, 0, path);              //(棋盘，已被使用的列，已被使用的45度对角线，已被使用的135度对角线，棋盘大小，当前行，该路径列坐标)
    for (auto path : result)                                    //每深入一层递归行数+1
    {
        for (int i = 0;i < path.size();i++)
        {
            cout << path[i] << ' ';
        }
        cout << endl;
    }
    return 0;
}
```

人狼羊过河

```
#include<iostream>
#include<vector>
#include<string>
#include<unordered_map>
using namespace std;
/*
MWGC->....
.W.C<-M.G.
MW.C->..G.
...C<-MWG.
M.GC->.W..
..G.<-MW.C
M.G.->.W.C
....<-MWGC
MWG.->...C
.W..<-M.GC
*/
vector<char> show{ 'M','W','G','C' };                       //用于按序遍历哈希表
unordered_map<string, int> situation;
void display(unordered_map<char, int>& man, unordered_map<char, int>& other, int count)   //count用于判断：奇数次为<-偶数次为->
{
    if (count % 2 == 0)                                     //偶数次过河，  ->
    {
        for (auto i : show)
        {
            if (man[i] != 0) { cout << i; }
            else { cout << '.'; }
        }
        cout << "->";
        for (auto i : show)
        {
            if (other[i] != 0) { cout << i; }
            else { cout << '.'; }
        }cout << endl;
    }
    else                                                    //奇数次过河，  <-
    {
        for (auto i : show)
        {
```

```
            if (other[i] != 0) { cout << i; }
            else { cout << '.'; }
        }
        cout << "<-";
        for (auto i : show)
        {
            if (man[i] != 0) { cout << i; }
            else { cout << '.'; }
        }cout << endl;
    }
}
void f(unordered_map<char, int>& man, unordered_map<char, int>& other, int
count)    //f(有人的岸，没人的岸，渡河次数)
{
    string sit = "";                                    //这里sit用于输出渡河的情况，用于判
断之前是否已经出现过该种情况，如果出现过则递归中止
    for (auto i : show)
    {
        if (man[i] != 0) { sit.push_back(i); }
        else { sit.push_back('.'); }
    }
    for (auto i : show)
    {
        if (other[i] != 0) { sit.push_back(i); }
        else { sit.push_back('.'); }
    }
    sit += to_string(count % 2);                            //最后加上count%2用于区分时
是MWGC->....(sit=MWGC....1) 和 ....<-MWGC(sit=MWGC....0) 因为这两种不是同一种情况
    situation[sit]++;

    if (situation[sit] > 1) {
        return;                         //中止情况：出现重复情况就中止该轮递归    注意
MWGC->....和 ....<-MWGC不是同一个情况，要注意区分
    }
    display(man, other, count);
    for (int i = 0;i < 4;i++)
    {
        //人单独过河，得保证狼和羊分别在两岸，  得保证菜和羊分别在两岸
        if (i == 0 && ((man['W'] == 1 && other['G'] == 1) || (man['G'] == 1 &&
other['W'] == 1)) && ((man['C'] == 1 && other['G'] == 1) || (man['G'] == 1 &&
other['C'] == 1)))
        {
            //cout<<1<<endl;
            man['M'] = 0;
            other['M'] = 1;
            f(other, man, count + 1);      //f(有人的岸，没人的岸，渡河次数)
            man['M'] = 1;
            other['M'] = 0;



        }//人狼过河  得保证菜和羊分别在两岸
        if (i == 1 && man['W'] == 1 && ((man['C'] == 1 && other['G'] == 1) ||
(man['G'] == 1 && other['C'] == 1)))
        {
```

```
            //cout<<2<<endl;
            man['M'] = 0;
            man['W'] = 0;
            other['M'] = 1;
            other['W'] = 1;
            f(other, man, count + 1);
            man['M'] = 1;
            man['W'] = 1;
            other['M'] = 0;
            other['W'] = 0;


        }//人菜过河， 得保证狼和羊分别在两岸，
        if (i == 2 && man['C'] == 1 && ((man['W'] == 1 && other['G'] == 1) ||
(man['G'] == 1 && other['W'] == 1)))
        {
            //cout<<3<<endl;
            man['M'] = 0;
            man['C'] = 0;
            other['M'] = 1;
            other['C'] = 1;
            f(other, man, count + 1);
            man['M'] = 1;
            man['C'] = 1;
            other['M'] = 0;
            other['C'] = 0;


        }//人羊过河
        if (i == 3 && man['G'] == 1)
        {
            //cout<<4<<endl;;
            man['M'] = 0;
            man['G'] = 0;
            other['M'] = 1;
            other['G'] = 1;
            f(other, man, count + 1);
            man['M'] = 1;
            man['G'] = 1;
            other['M'] = 0;
            other['G'] = 0;


        }
    }


}
int main()
{
    unordered_map<char, int> man;
    man['M'] = 1;
    man['W'] = 1;
    man['C'] = 1;
    man['G'] = 1;
```

```
    unordered_map<char, int> other;
    other['M'] = 0;
    other['W'] = 0;
    other['C'] = 0;
    other['G'] = 0;
    f(man, other, 0);


    return 0;
}
```

全排列

```
#include<iostream>
#include<vector>
#include<string>
#include<unordered_map>
using namespace std;
unordered_map<string, int> hash1;
void f(string& aaa)
{
    hash1[aaa]++;
    if (hash1[aaa] > 1)return;
    cout << aaa << endl;
    for (int i = 0;i < 3;i++)
    {
        for (int j = 0;j < 3;j++)
        {
            if (i == j)continue;
            if (aaa[i] != '\0' && aaa[j] != '\0')
            {
                string save = aaa;
                char temp = aaa[i];
                aaa[i] = aaa[j];
                aaa[j] = temp;
                f(aaa);
                aaa = save;
            }


        }
    }
}
int main()
{
    string aaa = "ABC";
    f(aaa);

}
```

岛屿数

```
#include<iostream>
#include<vector>
using namespace std;
/*
4 5
1 1 0 0 0
1 1 0 0 0
0 0 1 0 0
0 0 0 1 1
*/
void display(vector<vector<int>> aaa)
{
    cout << endl;
    for (auto xx : aaa)
    {
        for (int i = 0;i < xx.size();i++)
        {
            cout << xx[i] << ' ';
        }cout << endl;
    }
}



int result = 0;
int dir[4][2] = { {0, 1},{1, 0},{ -1, 0},{ 0, -1} };
void dfs(vector<vector<int>>& graph, vector<vector<int>>& visit, int x, int y)
{
    int next_x = 0;
    int next_y = 0;
    for (int i = 0;i < 4;i++)
    {
        next_x = x + dir[i][0];
        next_y = y + dir[i][1];
        if (next_x < 0 || next_x >= graph.size() || next_y < 0 || next_y >=
graph[0].size()) continue;  // 越界了，直接跳过
        if (visit[next_x][next_y] == 0 && graph[next_x][next_y] == 1)
        {
            visit[next_x][next_y] = 1;
            dfs(graph, visit, next_x, next_y);
            //visit[next_x][next_y]=0;     不需要回溯
            display(visit);
        }

    }
}

int main()
{

    int x, y;

    cout << "输入地图的行数x列数y: " << endl;
```

```
    cin >> x >> y;
    vector<vector<int>> graph;
    vector<vector<int>> visit(x, vector<int>(y, 0));
    cout << "输入地图，0为海洋1为岛屿：" << endl;
    for (int i = 0;i < x;i++)
    {
        vector<int> cow;
        for (int j = 0;j < y;j++)
        {
            int aaa;
            cin >> aaa;
            cow.push_back(aaa);
        }
        graph.push_back(cow);
    }
    int count = 0;
    for (int i = 0;i < x;i++)
    {
        for (int j = 0;j < y;j++)
        {
            if (visit[i][j] == 0 && graph[i][j] == 1)
            {
                visit[i][j] = 1;
                dfs(graph, visit, i, j);
                count++;
            }
        }

    }
    cout << "岛屿数量是：" << count << endl;
    display(visit);

    return 0;
}
```

走迷宫

```
#include<iostream>
#include<vector>
using namespace std;
//W表示墙    .表示路    *为输出从起点到终点的路径    o为输出尝试过的死路，输入地图和其实终点坐标，如果存在通路则输出，不存在则输出none
/*
5 7
W W . W W W W
W . . . W . .
W . W W W . W
W . . . W . W
W W W . W W W
0 2
4 3
输出
W W * W W W W
W * * o W . .
W * W W W . W
```

```
W * * * W . W
W W W * W W W
*/
/*
5 7
W W . W W W W
W . . . W . .
W . W W W . W
W . W . . . W
W W W . W W W
0 2
4 3
输出
None
*/
void display(vector<vector<char>> aaa)
{
    cout << endl;
    for (auto xx : aaa)
    {
        for (int i = 0;i < xx.size();i++)
        {
            cout << xx[i] << ' ';
        }cout << endl;
    }
}


int dir[4][2] = { {0, 1},{1, 0},{ -1, 0},{ 0, -1} };
              //上下左右四个方向
void dfs(vector<vector<char>>& graph, int x, int y, int endx, int endy)
                  //当前点坐标，终点坐标
{
    if (x == endx && y == endy)
                     //中止条件，即为当前点等于终点
    {
        return;
    }
    int next_x = 0;
    int next_y = 0;
    int count = 0;
    for (int i = 0;i < 4;i++)
                //递归上下左右四个方向
    {
        next_x = x + dir[i][0];
        next_y = y + dir[i][1];
        if (next_x < 0 || next_x >= graph.size() || next_y < 0 || next_y >=
graph[0].size()) continue;  // 越界了，直接跳过
        if (graph[next_x][next_y] == '.')
        {
            graph[next_x][next_y] = '*';
             //标记为路径
            dfs(graph, next_x, next_y, endx, endy);
            graph[next_x][next_y] == '.';
               //回溯
```

```
        }
        else {
 //count记录四个方向中没有路的数量
            count++;
        }
        if (count == 4)
 //如果4个方向都没有路，则标记当前位置为死路，即为'o'
        {
            graph[x][y] = 'o';
        }

    }
}

int main()
{

    int x, y, startx, starty, endx, endy;

    cout << "输入地图的行数x列数y: " << endl;
    cin >> x >> y;
    vector<vector<char>> graph;
    vector<vector<char>> visit(x, vector<char>(y, 0));
    cout << "输入地图.为路径W为墙壁: " << endl;
    for (int i = 0;i < x;i++)
    {
        vector<char> cow;
        for (int j = 0;j < y;j++)
        {
            char aaa;
            cin >> aaa;
            cow.push_back(aaa);
        }
        graph.push_back(cow);
    }
    cout << "输入起点坐标和终点坐标";
    cin >> startx >> starty >> endx >> endy;
    dfs(graph, startx, starty, endx, endy);                                    //因为是
一个连通域，所以不用for循环然后再dfs
    //像寻找岛屿数量是寻找连通域个数，需要用for(for)去找visit = 0的点
    if (graph[endx][endy] == '*')
    {
        display(graph);
    }
    else {
        cout << "NONE";
    }


    return 0;
}
```