```
******sc19l2w.patch******
test results
1. 2
2. 2
3. 2
4. 2
5. 2
6. 5
~~~~~~blocks.h~~~~~~
#ifndef BLOCKS_H
#define BLOCKS_H

// This class marks as object as blocking tom from walking here
class Blocks{};

#endif // BLOCKS_H
~~~~~~bomb.h~~~~~~
#ifndef BOMB_H
#define BOMB_H

#include "small.h"
#include "thing.h"

// a bomb that can be picked up.
class Bomb : public Thing, public Small
{
    string getName() { return "bomb"; }
};



#endif // BOMB_H
~~~~~~cave.cpp~~~~~~
#include <stdexcept>
#include <string>
#include <iostream>
#include <vector>
#include <string>

#include "cave.h"
#include "rock.h"
#include "thing.h"
#include "location.h"
#include "move.h"
#include "place.h"
#include "explode.h"
using namespace std;

Cave::Cave(int w, int h) : width(w), height(h) { // width and height of the cave

    /*if (width != 8 || height != 8)
        throw new logic_error("fixme: Cave needs to be fixed for non-standard sizes.");

    if ( width < 5 || height < 5)
        throw new logic_error("cave too small for tom.");*/


    map = new Location**[width];

    for (int x = 0; x < width; x++) {
        Location** column = new Location*[height];
        map[x] = column;
        for (int y = 0; y < height; y++)
            column[y] = new Location();

    }

    // create some rocks
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++)
            if (
                    (x == 0 || y == 0 || x == width-1 || y == height-1) )
                map[x][y] -> add( new Rock() );
    }

    tom = new Tom();

    // add tom to the middle of the map
    tom -> setLocation( this, width/2,height/2);
}



Cave::Cave(const Cave &r){

operator=(r);
```

```
}



void Cave::operator=(const Cave &r){
    width = r.width;
    height = r.height;


    map = new Location**[width];

    for (int x = 0; x < width; x++) {

        Location** column = new Location*[height];
            map[x] = column;
            for (int y = 0; y < height; y++)
                column[y] = new Location(*r.map[x][y]);
    }


    tom = new Tom();
    tom = r.tom;





}



Cave::~Cave() {
    //delete (map[0][0]); // fixme: I don't think this deletes all Locations and arrays declared in the constructor
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++)
            delete (map[x][y]);
    }





}

void Cave::command (string userCommand) {

    if (Move().triggersOn(userCommand))
        Move().fire(*this, userCommand);
    else if (Place().triggersOn(userCommand))
        Place().fire(*this, userCommand);
    else if (Throw().triggersOn(userCommand))
        Throw().fire(*this, userCommand);
    else if (Explode().triggersOn(userCommand))
        Explode().fire(*this, userCommand);
    else
        cerr << "tom doesn't know how to "+userCommand << endl;;
}

void Cave::show() {

    vector<Thing*>* thingsWithTom = map[tom -> getX()][tom -> getY()] -> getThings();

    for (int y = 0; y < height; y++) { // for all rows
        for (int x = 0; x < width; x++) // for all columns
            cout << map[x][y] -> show(); // output whatever we find there

        cout << "  "; // list the things at this location
        if (y <  (int) thingsWithTom -> size())
            cout << (*thingsWithTom)[y] -> getName();

        cout << endl;
    }

    cout << endl;
}
~~~~~~cave.h~~~~~~
#ifndef CAVE_H
#define CAVE_H

#include "location.h"
#include "tom.h"

using namespace std;

class Tom; // forward reference

// A cave which contains everything, including tom.
class Cave
```

```
{
public:
    Cave(){};
    Cave(int width, int height);

    Cave(const Cave &r);



    ~Cave();


    void operator=(const Cave &r);

    Location*** getMap() {return map;}
    void command (string s);
    void show();
    Tom *getTom() {return tom;}
    int getWidth() {return width;}
    int getHeight() {return height;}
private:
    int width, height;
    Tom *tom;
    Location*** map;
};

#endif // CAVE_H
~~~~~~CavePlusPlus.pro~~~~~~

CONFIG += c++11 console
#CONFIG -= app_bundle

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
        test.cpp \
        tom.cpp \
        cave.cpp \
        location.cpp \
        main.cpp

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

HEADERS += \
    blocks.h \
    bomb.h \
    cave.h \
    coin.h \
    command.h \
    explode.h \
    location.h \
    move.h \
    mushroom.h \
    place.h \
    rock.h \
    small.h \
    thing.h \
    tom.h
~~~~~~coin.h~~~~~~
#ifndef COIN_H
#define COIN_H

#include "small.h"
#include "thing.h"

// a small coin that can be picked up.
class Coin : public Thing, public Small
{
    string getName() { return "coin"; }
};

#endif // COIN_H
~~~~~~command.h~~~~~~
#ifndef COMMAND_H
#define COMMAND_H

#include "cave.h"

#include <string>

using namespace std;
```

```
// superclass of all user commands.
class Command {

public:
    Command(string triggerWord) : trigger(triggerWord) {}

    virtual bool triggersOn (string userCommand) {

        string userTrigger = userCommand;
        int index = userCommand.find(' ');
        if (index > 0)
                userTrigger = userCommand.substr(0, userCommand.find(' '));
        return trigger.compare(userTrigger) == 0;
    }

    virtual void fire(Cave& c, string arguments) =0;

protected:
    string tail(string userCommand) {

        int pos = userCommand.find(' ');
        if (pos >= 0)
            return userCommand.substr(userCommand.find(' ')+1, userCommand.length());
        else
            return "";
    }

private:
    string trigger;
};

#endif // COMMAND_H
~~~~~~explode.h~~~~~~
#ifndef EXPLODE_H
#define EXPLODE_H

#include "coin.h"
#include "command.h"
#include "mushroom.h"
#include "rock.h"
#include <iostream>
#include "bomb.h"
#include "thing.h"
#include <vector>
#include<iostream>
#include<algorithm>


void EXP(Cave &c, int a, int b){
    Location* locc= c.getMap()[a][b];
    vector<Thing*>* loc_c = locc -> getThings();
    Location* locn= c.getMap()[a][b-1];
    vector<Thing*>* loc_n = locn -> getThings();
    Location* locs= c.getMap()[a][b+1];
    vector<Thing*>* loc_s = locs -> getThings();







    //if(locc->isBomb())
    //{

        int y;
        int size;

        size = (int) loc_c->size();
        if(locc->isBomb())
        for (y = size-1; y >=0; y--)
        {
            if((*loc_c)[y]->getName()!="tom")
            {
                locc->remove((*loc_c)[y]);

            }

        }

        for (y = size-1; y >=0; y--)
        {
            if(b){

                if(locn->isBomb())
                    EXP(c,a,b-1);
```

```
        }

        if(a){
            Location* locw= c.getMap()[a-1][b];
            vector<Thing*>* loc_w = locw -> getThings();
            if(locw->isBomb())
            {
                EXP(c,a-1,b);}
        }



        if(b<c.getHeight()-1){
            if(locs->isBomb())
                EXP(c,a,b+1);
        }

        if(a<c.getWidth()-1){
            Location* loce= c.getMap()[a+1][b];
            vector<Thing*>* loc_e = loce -> getThings();
            if(loce->isBomb()){

                EXP(c,a+1,b);}
        }






    }

    /*if(b){
    size = (int) loc_n->size();
    for (y = size-1; y >=0; y--)
    {

        if(locn->isBomb())
            EXP(c,a,b-1);

        else if((*loc_n)[y]->getName()!="tom")
        {
            locn->remove((*loc_n)[y]);

        }


    }}

    if(a){
        Location* locw= c.getMap()[a-1][b];
        vector<Thing*>* loc_w = locw -> getThings();
    size = (int) loc_w->size();
    for (y = size-1; y >=0; y--)
    {

        if(locw->isBomb())
            EXP(c,a-1,b);

        if((*loc_w)[y]->getName()!="tom")
        {
            locw->remove((*loc_w)[y]);

        }


    }}


    if(b<c.getHeight()-1){
    size = (int) loc_s->size();
    for (y = size-1; y >=0; y--)
    {
        if(locs->isBomb())
            EXP(c,a,b+1);

        else if((*loc_s)[y]->getName()!="tom")
        {
            locs->remove((*loc_s)[y]);

        }


    }}


   if(a<c.getWidth()-1){
        Location* loce= c.getMap()[a+1][b];
```

```
            vector<Thing*>* loc_e = loce -> getThings();
        size = (int) loc_e->size();
        for (y = size-1; y >=0; y--)
        {
            if(loce->isBomb())
                EXP(c,a+1,b);

            else if((*loc_e)[y]->getName()!="tom")
            {
                loce->remove((*loc_e)[y]);

            }


        }} */


//}
    /*else
    {
        cerr << "There is no bomb"<< endl;
        return;}*/



}








class Explode : public Command {
public:
    Explode() : Command("explode") {};

    void fire(Cave& c, string userCommand) {

        int x = c.getTom()->getX();
        int y = c.getTom()->getY();
        EXP(c,x,y);


    }
};





#endif // EXPLODE_H
~~~~~~location.cpp~~~~~~

#include <algorithm>

#include "blocks.h"
#include "location.h"
#include "small.h"
#include "tom.h"
#include "bomb.h"
#include "coin.h"
#include "mushroom.h"
#include "rock.h"
using namespace std;

void Location::add(Thing *t) {

    things.push_back( t );
}

void Location::remove(Thing *t) {

    // note we don't free the memory here
    things.erase(std::remove(things.begin(), things.end(), t), things.end());
}


char Location::show() {

    int pickUp = 0, blocking = 0, tom = 0;

    // count properties of things at this location
    for (Thing * t : things) {
        if (dynamic_cast<Blocks*>(t) ) // is t subclass of Blocks?
            blocking++;
        if (dynamic_cast<Small*>(t)) // is t subclass of Pickable?
            pickUp++;
```

```
        if (dynamic_cast<Tom*> (t) != NULL )
            tom++;
    }

    // return a character based on the properties
    if (blocking) // remember that
        return 'X';
    else if (pickUp > 0) {
        if (tom)
            return 'L';
        else
            return '_';

    }
    else if (tom)
        return '|';
    else
        return '.';

}

Location::Location(const Location &loc){
    for (Thing * t : loc.things) {
        if (dynamic_cast<Coin*>(t) )
            add(new Coin());
        if (dynamic_cast<Mushroom*>(t))
            add(new Mushroom());
        if (dynamic_cast<Bomb*> (t) )
            add(new Bomb());
        if (dynamic_cast<Rock*> (t) )
            add(new Rock());
    }

       ++count;

}



bool Location::isBlocking() { // does this location block tom's travels?

    int blocking = 0;

    for (Thing * t : things)
        if ( dynamic_cast<Blocks*>(t) )// is t subclass of Blocks?
            blocking++;

    return blocking; // 0 means false, otherwise true
}

bool Location::isBomb() { // does this location block tom's travels?

    int bombing = 0;

    for (Thing * t : things)
        if ( dynamic_cast<Bomb*>(t) )// is t subclass of Blocks?
            bombing++;

    return bombing; // 0 means false, otherwise true
}

bool Location::isCoin() { // does this location block tom's travels?

    int coining = 0;

    for (Thing * t : things)
        if ( dynamic_cast<Bomb*>(t) )// is t subclass of Blocks?
            coining++;

    return coining; // 0 means false, otherwise true
}

bool Location::isMushroom() { // does this location block tom's travels?

    int mushrooming = 0;

    for (Thing * t : things)
        if ( dynamic_cast<Bomb*>(t) )// is t subclass of Blocks?
            mushrooming++;

    return mushrooming; // 0 means false, otherwise true
}

~~~~~~location.h~~~~~~
#ifndef LOCATION_H
#define LOCATION_H
#include <iostream>
```

```
#include <vector>

#include "thing.h"

class Location
{
public:
    Location() { ++count; }

    Location(const Location& loc);

    ~Location() {
        for (auto t : things)
            delete(t);
        --count;
    }

    void add(Thing *t);
    void remove(Thing *t);
    char show();
    bool isBlocking();
    bool isBomb();
    bool isCoin();
    bool isMushroom();
    vector<Thing*>* getThings()
    { return &things; }
    static int count; // a count of the number of locations allocated/deleted


private:

    vector<Thing*> things;

};









#endif // LOCATION_H
~~~~~~main.cpp~~~~~~
/***
 * Do not change this file - but you might edit the arguments passed to main:
 *  - Projects (spanner on right)
 *  - Under "Build and Run", select "Run" (small green arrow)
 *  - Edit "Command line arguments" (main panel)
 *
 * v1: initial release
 * v2: removed std:: from some strings & allowed students to edit the Location class
 */

#include <string>
#include <iostream>

#include "cave.h"
#include "test.cpp"

using namespace std;


int main(int argc, char** argv) {

    int width = 8, height = 8; // provided code only works for these values(!)

    if (argc == 3) {
        width  = stoi (argv[1]);
        height = stoi (argv[2]);
    }

    if (argc != 2) {

        Cave c(width, height);

        string input;
        while (true) {

            c.show();

            cout << ">";
            getline(cin, input);

            if (input.compare("exit") == 0)
                break;
```

```
            c.command(input);
        }
    }
    else test();
}
~~~~~~move.h~~~~~~
#ifndef MOVE_H
#define MOVE_H

#include "cave.h"
#include "command.h"

#include <string>
#include <iostream>
#include <string>

using namespace std;

// a command to move tom around the cave.
class Move : public Command
{
public:
    Move() : Command("move") {};

    void fire(Cave& c, string userCommand) {

        string s = tail(userCommand);

        int newTomX = c.getTom()->getX(),  newTomY = c.getTom()->getY();

        if (s[0] == 'w') //west
            newTomX--;
        else if  (s[0] == 'n') //north
            newTomY--;
        else if  (s[0] == 'e') //east
            newTomX++;
        else if  (s[0] == 's') //south
            newTomY++;
        else {
            cerr << "tom can't \"" << s << "\"" << endl;
            return;
        }


        if (newTomX < 0 || newTomY < 0 || newTomX >= c.getHeight() ||newTomY >= c.getHeight())
        {
            cerr << "can't walk into the void" << endl;
            return;
        }
        else if ( c.getMap()[newTomX][newTomY] -> isBlocking() )
        {
            cerr << "something is blocking the way" << endl;
            return;
        }
        else if (newTomX != c.getTom()->getX() || newTomY != c.getTom()->getY())
        {
            c.getTom() -> setLocation (&c, newTomX, newTomY);
            cerr << "tom shuffles through the dank cave" << endl;
            return;
        }
    }
};

#endif // MOVE_H
~~~~~~mushroom.h~~~~~~
#ifndef MUSHROOM_H
#define MUSHROOM_H

#include "small.h"
#include "thing.h"

// a small glowing mushroom on the ground, found wherever adventurers wander. Can be picked up.
class Mushroom : public Thing, public Small
{
    string getName() { return "mushroom"; }
};

#endif // MUSHROOM_H
~~~~~~place.h~~~~~~
#ifndef PLACE_H
#define PLACE_H

#include "coin.h"
#include "command.h"
#include "mushroom.h"
#include "rock.h"
#include <iostream>
```

```
#include "bomb.h"
using namespace std;

class Place : public Command {
public:
    Place() : Command("place") {};

    void fire(Cave& c, string userCommand) {

        string s = tail(userCommand);
        Location* loc = c.getMap()[c.getTom()->getX()][c.getTom()->getY()];

        /*for(int a=0; a<8; a++){
            for(int b=0; b<8; b++){
                Location* loc1 = c.getMap()[a][b];
                loc1 -> add(new Bomb());
            }
        } */


        if (s.compare("coin")==0)
            loc -> add(new Coin());
        else if (s.compare("mushroom")==0)
            loc -> add(new Mushroom());
        else if (s.compare("bomb")==0)
            loc -> add(new Bomb());
        else
            cerr << "I don't know how to place a " << userCommand << endl;
    }
};


class Throw : public Command {
public:
    Throw() : Command("throw") {};

    void fire(Cave& c, string userCommand) {

        string s = tail(userCommand);

        if (s.compare("coin west")==0)
        {   Location* loc = c.getMap()[c.getTom()->getX()-1][c.getTom()->getY()];
            if ( loc -> isBlocking() )
            {
                cerr << "something is blocking the way" << endl;
                return;
            }
            loc -> add(new Coin());
        }
        else if (s.compare("coin east")==0)
        {   Location* loc = c.getMap()[c.getTom()->getX()+1][c.getTom()->getY()];
            if ( loc -> isBlocking() )
            {
                cerr << "something is blocking the way" << endl;
                return;
            }
            loc -> add(new Coin());
        }
        else if (s.compare("coin north")==0)
        {   Location* loc = c.getMap()[c.getTom()->getX()][c.getTom()->getY()-1];
            if ( loc -> isBlocking() )
            {
                cerr << "something is blocking the way" << endl;
                return;
            }
            loc -> add(new Coin());
        }
        else if (s.compare("coin south")==0)
        {   Location* loc = c.getMap()[c.getTom()->getX()][c.getTom()->getY()+1];
            if ( loc -> isBlocking() )
            {
                cerr << "something is blocking the way" << endl;
                return;
            }
            loc -> add(new Coin());

        }
        else
            cerr << "I don't know how to throw a " << userCommand << endl;
    }
};
```

```
#endif // PLACE_H
~~~~~~rock.h~~~~~~
#ifndef ROCK_H
#define ROCK_H

#include "blocks.h"
#include "thing.h"
#include <string>

// a giant bolder that stops tom from walking through this location
class Rock: public Thing, public Blocks
{
public:
    Rock(){};
    ~Rock(){};
    string getName() {return "rock";}
};

#endif // ROCK_H
~~~~~~small.h~~~~~~
#ifndef SMALL_H
#define SMALL_H

// This class marks an object as being small
class Small{};

#endif // SMALL_H
~~~~~~test.cpp~~~~~~
/***
 * Do not change this file
 *
 * A script very much like this will be used to grade your code.
 *
 * I may add additional checks to ensure that no cheating takes place!
 *
 */

#include <iostream>
#include <fstream>
#include <stdexcept>
#include <sstream>
#include<iostream>
#include<string>
#include <dirent.h> // this might be tricky, but mostly there for gnu compilers.

#include "cave.h"
#include "coin.h"
#include "location.h"
#include "mushroom.h"


using namespace std;

int Location::count;
int Thing::count;

int test1() {

    Cave c(8,8);
    c.getTom()->setLocation(&c, 5, 5);

    bool goodA = true;

    // let's walk in a circle and check we get the results we expect
    c.command("move west");
    goodA &= (c.getTom()->getX() == 4);

    c.command("move north");
    goodA &= (c.getTom()->getY() == 4);

    c.command("move east");
    goodA &= (c.getTom()->getX() == 5);

    c.command("move south");
    goodA &= (c.getTom()->getY() == 5);

    // walk into wall to east
    for (int i = 0; i < 10; i++)
        c.command("move east");

    bool goodB = true;
    goodB &= (c.getTom()->getX() == 6);
    goodB &= (c.getTom()->getY() == 5);

    return (goodA ? 1 : 0) + (goodB ? 1 : 0);
}
```

```
int test2() {

    bool goodA = true, goodB = true;

    try {

        for (int i = 5; i < 20; i+=2)
            for (int j = 5; j < 27; j+=3) {
                Cave c(i,j);

                goodA &= c.getWidth() == i;
                goodA &= c.getHeight() == j;

                for (int x = 0; x < i; x++)
                    for (int y = 0; y < j; y++)
                        if (x > 0 && x < i-1 && y> 0 && y < j-1)
                            goodB &= !c.getMap()[x][y]->isBlocking();
                        else
                            goodB &= c.getMap()[x][y]->isBlocking();
            }

    }
    catch (...) // an errors (including logic_error) fall through here
    {
        goodA = goodB = false;
    }

    return (goodA ? 1 : 0) + (goodB ? 1 : 0);
}

int test3() {

    Location::count = 0; // reset the counters in location and thing
    Thing::count = 0;

    bool goodA = true;
    {
        int x = 8, y= 8;

        Cave c(x, y);

        goodA &= Location::count == x*y;
    }

    goodA &= Location::count == 0;
    goodA &= Thing::count == 0;

    return goodA ? 2 : 0;
}

void test4CheckPointers(Cave *a, Cave &b, bool& goodA) {

    goodA &= b.getMap() != a -> getMap(); // check that we created a new map
    goodA &= b.getMap()[0][0] != a -> getMap()[0][0]; // has the vector in Location been copied?
    goodA &= b.getMap()[0][0]->getThings()->at(0) != a -> getMap()[0][0]->getThings()->at(0); // has the Rock been copied
}

int test4() {

    bool goodA = true, goodB = true;

    Cave *a = new Cave(8, 8); // so we don't destroy any (accidently) shallow copied copies twice
    Cave b(*a);

    test4CheckPointers(a, b, goodA);

    Cave *c = new Cave(8,8);
    Cave d(8,8);
    d = *c;

    test4CheckPointers(c, d, goodB);

    return (goodA ? 1 : 0) + (goodB ? 1 : 0);
}

bool hasCoin (Cave &c, int x, int y) {

    for (auto t : *c.getMap()[x][y]->getThings())
        if ( dynamic_cast<Coin*>(t) )
            return true;

    return false;
}

bool hasMushroom (Cave &c, int x, int y) {
    for (auto t : *c.getMap()[x][y]->getThings())
        if ( dynamic_cast<Mushroom*>(t) )
            return true;

    return false;
}

int test5() {

    bool goodA = true, goodB = true;

    Cave c(8,8);
    c.getTom()->setLocation(&c, 5, 5);
    c.command("throw coin north");
    goodA &= hasCoin(c, 5,4);

    c.command("throw coin east");
    goodA &= hasCoin(c, 6,5);

    c.getTom()->setLocation(&c, 6,6);

    c.command("throw coin west");
    goodA &= hasCoin(c, 5,6);

    c.command("throw coin east");
    goodB &= !hasCoin(c, 7,6); // can't throw - rock
    goodB &= goodA; // no marks for rock-blocking if throwing didn't work

    return (goodA ? 1 : 0) + (goodB ? 1 : 0);
}


bool hasBomb (Cave &c, int x, int y) {

    for (auto t : *c.getMap()[x][y]->getThings())
        if ( t->getName().compare("bomb") == 0 )
            return true;

    return false;
}

int countBombs(Cave *c) {
    int count = 0;
    for (int x = 0; x < c->getWidth(); x++)
        for (int y = 0; y < c->getHeight(); y++)
            if (hasBomb(*c, x, y))
                count++;

    return count;
}

int test6() {

    bool goodA = true, goodB = true, goodC = true;

    Cave c(8,8);


    goodA &= countBombs(&c) == 0;

    const int bombCount = 8;
    int bombs[bombCount][2] = {{1,1},{2,2},{1,7},{6,6},{5,5},{6,5},{5,6},{7,5}};

    for (int x = 0; x < bombCount; x++) {
        c.getTom()->setLocation(&c,bombs[x][0],bombs[x][1]);
        c.command("place bomb");
    }

    goodA &= countBombs(&c) == bombCount;

    c.getTom()->setLocation(&c,5,5);
    c.command("place mushroom");
    c.getTom()->setLocation(&c,4,5);
    c.command("place mushroom");

    c.getTom()->setLocation(&c,1,2);
    c.command("explode");

    goodB &= countBombs(&c) == 6;

    c.getTom()->setLocation(&c,5,4);
    c.command("explode");
    goodB &= countBombs(&c) == 1;

    goodC &= !c.getMap()[7][5]->isBlocking(); // a bomb inside the rock should destroy the rock
    goodC &=  c.getMap()[7][6]->isBlocking(); // other rocks remain untouched
    goodC &= !hasMushroom (c, 5,5); // mushroom should be destroyed
    goodC &=  hasMushroom (c, 4,5); // mushroom should not explode

    c.command("place bomb");

    return (goodA ? 1 : 0) + (goodB ? 3 : 0) + (goodC ? 1 : 0);
}

bool endsWith(string const & value, string const & ending) {

    if (ending.size() > value.size()) return false;
    return equal(ending.rbegin(), ending.rend(), value.rbegin());
}

bool isEmpty(const string& file)  { // https://stackoverflow.com/questions/2424138/portable-way-to-get-file-size-in-c-c

    ifstream ifile(file);
    ifile.seekg(0, ios_base::end);
    return ifile.tellg() == 0;
}

void test() {

    cerr.setstate(ios_base::failbit); // no error output while testing please!

    cout << "Enter folder containing all cpp source files (or enter to only run tests):";
    string folder;
    getline(cin, folder); // comment out this line to test quickly

    stringstream buffer;
    string username = "";

   if (folder.length() > 0) {
        cout << endl <<"Enter leeds username (sc19xxx):";
        getline(cin, username);
        username += ".patch";

        if (auto dir = opendir(folder.c_str())) {
            while (auto f = readdir(dir)) {
                if (!f->d_name || f->d_name[0] == '.')
                    continue;

                string name = string (f->d_name);
                if (endsWith(name, ".cpp") || endsWith(name, ".pro") || endsWith(name, ".h") ) {
                    printf("Adding file: %s\n", f->d_name);

                    string fileName = folder+"/"+name;

                    if (isEmpty(fileName)) {
                        cout <<"...warning - empty file, please remove" << endl;
                        continue;
                    }

                    ifstream file(fileName );
                    stringstream tmp;

                    buffer << "~~~~~~"<<name<<"~~~~~~\n";
                    buffer << file.rdbuf();
                }
            }
            closedir(dir);
        }
        else {
            cout <<"folder not found: " << folder << endl;
            return;
        }
    }

    stringstream testResults;

    testResults << "test results" << endl;
    testResults << "1. " << test1() << endl;
    testResults << "2. " << test2() << endl;
    testResults << "3. " << test3() << endl;
    testResults << "4. " << test4() << endl;
    testResults << "5. " << test5() << endl;
    testResults << "6. " << test6() << endl;

    cout << testResults.str();

    if (folder.length() > 0) {
        ofstream outfile;
        outfile.open(folder+"/"+username, ios::out | ios::trunc );
        outfile << "******" << username << "******\n";
        outfile << testResults.str();
        outfile << buffer.str();
        outfile.close();
    }
```

```
}
~~~~~~thing.h~~~~~~
/***
 * Do not change this file
 */

#ifndef THING_H
#define THING_H

#include <string>
using namespace std;

// the superclass of all things in the cave.
class Thing
{
public:
    Thing(){++count;};
    virtual ~Thing(){--count;};
    virtual string getName() =0;
    static int count; // a count of the number of locations allocated/deleted
};

#endif // THING_H
~~~~~~tom.cpp~~~~~~
/***
 * Do not change this file
 */

#include "tom.h"
#include "cave.h"

void Tom::setLocation(Cave* c, int new_x, int new_y) {

    if (location[0] >= 0 && location[1] >= 0)  // if we've set the location before, remove tom from old position
        c -> getMap()[location[0]][location[1]]->remove(this);

    location[0] = new_x;
    location[1] = new_y;

    c -> getMap()[location[0]][location[1]]->add(this);
}
~~~~~~tom.h~~~~~~
/***
 * Do not change this file
 */

#ifndef TOM_H
#define TOM_H

#include "cave.h"
#include "thing.h"

class Cave; // forward reference because the tom class needs to know about the cave class
class Tom : public Thing
{
public:
    Tom(){}
    ~Tom(){}
    string getName() {return "tom";}
    void setLocation(Cave* c, int x, int y);
    int getX() {return location[0];}
    int getY() {return location[1];}
private:
    int location[2] = {-1,-1}; // fixed length array, so we can allocate automatically...
                               // ...note invalid location (-1,-1) before setLocation call.
};

#endif // TOM_H
```