```cpp
// TreasureHunt.cpp
// Assignment 1.2 By Nathan Graham

#include "stdafx.h"
#include <iostream>
#include <string>
using namespace std;

const int arraySize = 10; // size of the array of rooms
const int playerInventorySize = 5; // number of spaces in the players inventory
const int roomInventorySize = 50; // number of items that can be in each room
const string endLine = "\n"; // makes the end lines more organised

enum wallDirection { horizontal = 0, vertical = 1 }; // horizontal and vertical walls which create a grid over the array of rooms
enum door { noDoor = 0, lockedDoor = 1, openDoor = 2, corridor = 3, stairCase = 4, forest = 5, openArea = 6, tunnel = 7}; //types of walls that can exist on the grid
enum inventory { emptyItem = 0, skeletonKey = 1, lantern = 2, litLantern = 3, food = 4, waterBottle = 5, matches = 6, goldKey = 7 }; //items which can be pickedup into the inventory from each room


class inventoryList {
public:
    inventory inventoryType;
    string description;
    inventoryList() { inventoryType = inventory::emptyItem, description = "Empty"; };
    inventoryList(inventory a, string b ) { inventoryType = a, description = b; };
};

class wall { // class for the walls which have no doors or openings
public:
    door doorType;
    wall() { doorType = noDoor; };
};

class room { // class for each room in the array
public:
    string description;
    wall *northWall;
    wall *southWall;
    wall *eastWall;
    wall *westWall;
    bool visible;
    inventory enableVisible; // allows the player to turn visibility on, when a room is to dark
    inventory enableUnlock; // allows the player to unlock a door, when the door to next room is locked
    bool treasureRoom;
    inventory slots[roomInventorySize];
    room() { description = "Empty room", visible = true, enableVisible = inventory::emptyItem, enableUnlock = inventory::emptyItem, treasureRoom = false; }; // default options for each room

    void updateRoom(string name, door northDoor, door southDoor, door eastDoor, door westDoor, bool roomVisible) { // function which is used for creating doors in walls for each room on the array
        description = name;
        northWall->doorType = northDoor;
        southWall->doorType = southDoor;
        eastWall->doorType = eastDoor;
        westWall->doorType = westDoor;
        visible = roomVisible;
    }

    bool addInventory(inventory inventoryItem) { // function which returns true for adding inventoryItems to each room if they are the set ones and false for anything else
        for (int i = 0; i < roomInventorySize; i++) {
            if (slots[i] == inventory::emptyItem) {
                slots[i] = inventoryItem;
                return true;
            }
        }

        return false;
    }

    bool removeInventory(inventory inventoryItem) { // function which returns true for removing inventoryItems which can only be used once and false for anything else
        for (int i = 0; i < roomInventorySize; i++) {
            if (slots[i] == inventoryItem) {
                slots[i] = inventory::emptyItem;
                return true;
            }
        }

        return false;
    }

    bool unlockDoor() { // allows the player to unlock the locked doors that connect to another room in each direction
        bool unlockedDoor = false;

        if (northWall->doorType == door::lockedDoor) {
            northWall->doorType = door::openDoor;
            unlockedDoor = true;
        }

        if (southWall->doorType == door::lockedDoor) {
            southWall->doorType = door::openDoor;
            unlockedDoor = true;
        }

        if (eastWall->doorType == door::lockedDoor) {
            eastWall->doorType = door::openDoor;
            unlockedDoor = true;
        }

        if (westWall->doorType == door::lockedDoor) {
            westWall->doorType = door::openDoor;
            unlockedDoor = true;
        }

        return unlockedDoor;
    }
};

class player { // this is the player description, with commands they can do for moving around the array, adding and removing items from
public:        // inventory and for eating food to maintain energy levels which are reduced each time the player moves room
    string name;
    int x;
    int y;
    int energy;
    inventory slots[playerInventorySize];
    player() { name = "Link:", x = 4, y = 4, energy = 30; };

    bool moveNorth(door doorType) {
        if (y > 0
            && doorType != lockedDoor
            && doorType != noDoor) {
            y--;
            energy--;
            return true;
        }
        return false;
    }

    bool moveSouth(door doorType) {
        if (y < (arraySize - 1)
            && doorType != lockedDoor
            && doorType != noDoor) {
            y++;
            energy--;
            return true;
        }
        return false;
    }

    bool moveEast(door doorType) {
```

```cpp
        if (x < (arraySize - 1)
            && doorType != lockedDoor
            && doorType != noDoor) {
            x++;
            energy--;
            return true;
        }
        return false;
    }

    bool moveWest(door doorType) {
        if (x > 0
            && doorType != lockedDoor
            && doorType != noDoor) {
            x--;
            energy--;
            return true;
        }
        return false;
    }

    bool addInventory(inventory inventoryItem) {
        for (int i = 0; i < playerInventorySize; i++) {
            if (slots[i] == inventory::emptyItem) {
                slots[i] = inventoryItem;
                return true;
            }
        }

        return false;
    }

    bool removeInventory(inventory inventoryItem) {
        for (int i = 0; i < playerInventorySize; i++) {
            if (slots[i] == inventoryItem) {
                slots[i] = inventory::emptyItem;
                return true;
            }
        }

        return false;
    }

    bool eatFoods() {
        if (energy >= 30)
            return false;

        energy += 5;

        if (energy > 30) {
            energy = 30;
        }

        return true;
    }
};

void displayDirection(door doorType, string direction) // function which displays
   the direction of the door type
{
    switch (doorType) // used to determine which doortype should be used
    {
        case door::lockedDoor:
            cout << direction << " there is a locked door." << endLine;
            break;
        case door::openDoor:
            cout << direction << " there is an open door." << endLine;
            break;
        case door::corridor:
            cout << direction << " there is a corridor." << endLine;
            break;
        case door::stairCase:
            cout << direction << " there is a staircase." << endLine;
            break;
        case door::forest:
            cout << direction << " there is a forest." << endLine;
            break;
        case door::openArea:
            cout << direction << " there is an open area." << endLine;
            break;
        case door::tunnel:
            cout << direction << " there is a tunnel." << endLine;
            break;
    }
}

void displayRoom(room *currentRoom, player *currentPlayer) // function which
  displays things about the current room and the player
{
    // initialise inventory

    inventoryList availableItems[]{
        inventoryList(inventory::emptyItem, "Empty"),
        inventoryList(inventory::skeletonKey, "Bunch of keys including a skeleton
           key"),
        inventoryList(inventory::lantern, "Lantern"),
        inventoryList(inventory::litLantern, "Lit Lantern"),
        inventoryList(inventory::food, "An array of delicious cakes and drinks"),
        inventoryList(inventory::waterBottle, "Water bottle"),
        inventoryList(inventory::matches, "A box of matches"),
        inventoryList(inventory::goldKey, "A large gold key. This looks very
          old."),
    };

    cout << currentPlayer->name << " Current Energy: " << currentPlayer->energy << endLine << "Your location is " << currentRoom->description << endLine; // displays current energy levels, current rooms
                                                                      // description and current items in inventory
    int inventorySize = (sizeof(availableItems) / sizeof(*(availableItems))); // size of array divided by size of element

    bool overrideDarkness = false; // statement to say each room is not overrided by darkness by default

    if (!currentRoom->visible) { // If statement saying that if the room is not visible then it must be overrided by darkness
        for (int i = 0; i < playerInventorySize; i++) {
            if (currentPlayer->slots[i] == currentRoom->enableVisible) {
                overrideDarkness = true;
                break;
            }
        }
    }

    if (currentRoom->visible
        || overrideDarkness) { // If statement saying that if the room is visible, and not overrided by darkness, then display the directions for each doorType
        displayDirection(currentRoom->northWall->doorType, "To the north");
        displayDirection(currentRoom->southWall->doorType, "To the south");
        displayDirection(currentRoom->eastWall->doorType, "To the east");
        displayDirection(currentRoom->westWall->doorType, "To the west");
        bool titleDisplayed = false;

        for (int i = 0; i < roomInventorySize; i++) { // If the room inventory is not empty, display what items are in the room
            if (currentRoom->slots[i] != inventory::emptyItem) {

                if (!titleDisplayed) {
                    cout << "Room item(s) include:" << endLine;
                    titleDisplayed = true;
                }

                for (int s = 0; s < inventorySize; s++) {

                    if (currentRoom->slots[i] == availableItems[s].inventoryType) { // If current room has enough slots for available items in the room inventory, give the available items a slot number e.g [0]
                        cout << "\t" << availableItems[s].description << "[" << i << "]" << endLine; // '\t' is done to indent the line
                        break;
                    }
                }
            }
        }
    }
    else
        cout << "Everything is dark. You can hear something scurrying around your feet." << endLine; //if room is not visible, then output this text

    bool titleDisplayed = false; // set as default for items in inventory to not be displayed

    for (int i = 0; i < playerInventorySize; i++) {  // for the players inventory size if there is an item in inventory, then display items in inventory, then for items in inventory, give them a slot number
        if (currentPlayer->slots[i] != inventory::emptyItem) {

            if (!titleDisplayed) {
                cout << "Your inventory includes:" << endLine;
                titleDisplayed = true;
            }

            for (int s = 0; s < inventorySize; s++) {

                if (currentPlayer->slots[i] == availableItems[s].inventoryType) {
                    cout << "\t" << availableItems[s].description << "[" << i << "]" << endLine;
                    break;
                }
            }
        }
    }

    if (currentPlayer->energy < 5) // if the players energy levels reach 5, this will be displayed
        cout << ">>> You are feeling really ill. Maybe you should eat something?" << endLine;
    else                          // if the players energy levels reach 10, this will be displayed
        if (currentPlayer->energy < 10)
            cout << ">>> You are feeling weak. Maybe you should eat something?" << endLine;
}

int main()
{
    wall walls[2][arraySize + 1][arraySize];  // [horizontal/vertical] [Lines + 1] [walls with a line]
    room rooms[arraySize][arraySize]; // [x coordinate] [y coordinate]

    // initialsie rooms with walls and defaults and empty inventory
    for (int x = 0; x < arraySize; x++) {
        for (int y = 0; y < arraySize; y++) {
            rooms[x][y].northWall = &walls[horizontal][y][x];
            rooms[x][y].southWall = &walls[horizontal][y + 1][x];
            rooms[x][y].eastWall = &walls[vertical][x + 1][y];
            rooms[x][y].westWall = &walls[vertical][x][y];

            for (int i = 0; i < roomInventorySize; i++) {
                rooms[x][y].slots[i] = inventory::emptyItem;
            }
        }
    }

    // initialise player
    player currentPlayer;

    for (int i = 0; i < playerInventorySize; i++)
        currentPlayer.slots[i] = inventory::emptyItem;

    currentPlayer.addInventory(inventory::matches);

    // defined rooms

    rooms[4][4].updateRoom("Home" + endLine + "There is a log fire burning" + endLine + "You can smell coffee brewing and everything seems normal", door::openDoor, door::openDoor, door::corridor, door::noDoor, true);
    rooms[4][4].addInventory(inventory::lantern);
    rooms[4][3].description = "in a field of grass" + endLine + "Mountains can be spotted from in the distance";
    rooms[4][3].northWall->doorType = openArea;
    rooms[4][2].description = "the footsteps of a mountain pathway" + endLine + "There is also a mud trail leading into a forest to the west";
    rooms[4][2].northWall->doorType = stairCase;
    rooms[4][2].westWall->doorType = forest;
    rooms[4][2].addInventory(inventory::waterBottle);
    rooms[3][2].description = "inside the forest surrounded by tall trees" + endLine + "The light being blocked out by them";
    rooms[3][2].westWall->doorType = forest;
    rooms[2][2].description = "a forest which is pitch black";
    rooms[2][2].visible = false;
    rooms[2][2].enableVisible = inventory::litLantern;
    rooms[2][2].addInventory(inventory::skeletonKey);
    rooms[4][1].description = "haendLine way up the mountain pass";
    rooms[4][1].northWall->doorType = stairCase;
    rooms[4][0].description = "at the mountains summit" + endLine + "From here you have a great view of the forest and the fields" + endLine + "As well as the house you came out off";
    rooms[4][0].addInventory(inventory::food);

    rooms[5][4].description = "a long corridor" + endLine + "There is a family portrait on the wall, it looks a lot like you.";
    rooms[5][4].eastWall->doorType = door::openDoor;
    rooms[6][4].description = "You have found your way into a larder" + endLine + "This must be used be storing some food";
    rooms[6][4].visible = false;
    rooms[2][2].enableVisible = inventory::litLantern;
    rooms[6][4].addInventory(inventory::food);

    rooms[4][5].description = "a staircase" + endLine + "Your uncle used to go down there to work" + endLine + "back when he was still here";
    rooms[4][5].southWall->doorType = stairCase;
    rooms[4][6].description = "a hall at the bottom of the stairs";
    rooms[4][6].southWall->doorType = lockedDoor;
    rooms[4][6].enableUnlock = inventory::skeletonKey;
    rooms[4][7].description = "an empty room with a painting of a dog on the wall" + endLine + "You don't remember owning a dog";
    rooms[4][7].eastWall->doorType = openDoor;
    rooms[4][7].westWall->doorType = openDoor;

    rooms[5][7].description = "a long corridor" + endLine + "You can't quite see the end";
    rooms[5][7].eastWall->doorType = corridor;
    rooms[6][7].description = "The long corridor" + endLine + "Torches on the wall are lighting the way towards the end";
    rooms[6][7].eastWall->doorType = corridor;
    rooms[7][7].description = "the end of the corridor" + endLine + "The skeleton of a small bird is resting on the floor";
    rooms[7][7].southWall->doorType = stairCase;
    rooms[7][7].addInventory(inventory::matches);
    rooms[7][8].description = "a labortory" + endLine + "Uncle must have done his work in here";
    rooms[7][8].eastWall->doorType = openDoor;
    rooms[7][8].visible = false;
    rooms[2][2].enableVisible = inventory::litLantern;
    rooms[8][8].description = "a test chamber" + endLine + "It's a mess" + endLine + "looks like a lot of experiments failed in here";
    rooms[8][8].eastWall->doorType = openDoor;
    rooms[8][8].visible = false;
    rooms[2][2].enableVisible = inventory::litLantern;
    rooms[9][8].description = "a storage room" + endLine + "Seems like there are all kinds of things stored here" + endLine + "From chemical formulas to rations of food";
    rooms[9][8].visible = false;
    rooms[2][2].enableVisible = inventory::litLantern;
    rooms[9][8].addInventory(inventory::food);

    rooms[3][7].description = "a natural cave" + endLine + "Enough space for you to crouch and walk around";
    rooms[3][7].westWall->doorType = tunnel;
    rooms[2][7].description = "an enclosed area" + endLine + "Just enough head space for you to crawl around" + endLine + "The cave seems to get tighter the futher in you go";
    rooms[2][7].westWall->doorType = tunnel;
    rooms[2][7].southWall->doorType = stairCase;
    rooms[1][7].description = "an open area" + endLine + "After squeasing through the tunnel you arrived at a door";
    rooms[1][7].westWall->doorType = openDoor;
    rooms[0][7].description = "a singuar room with a opened box";
    rooms[0][7].addInventory(inventory::food);
    rooms[0][7].addInventory(inventory::goldKey);
    rooms[2][8].description = "a dark area" + endLine + "A map of the underground system is on the wall";
    rooms[2][8].eastWall->doorType = openArea;
    rooms[2][8].visible = false;
    rooms[2][2].enableVisible = inventory::litLantern;
    rooms[3][8].description = "a thin corridor" + endLine + "There is a large vault at the end";
    rooms[3][8].southWall->doorType = lockedDoor;
    rooms[3][8].enableUnlock = inventory::goldKey;
    rooms[3][8].visible = false;
    rooms[2][2].enableVisible = inventory::litLantern;
    rooms[3][9].description = "inside the vault" + endLine + "You have found your family's lost treasures!";
    rooms[3][9].treasureRoom = true;

    while (true) {
        room *currentRoom = &(rooms[currentPlayer.x][currentPlayer.y]); // Pointer to the room in the array
        displayRoom(currentRoom, &currentPlayer);

        if (currentPlayer.energy <= 0) { // if your energy reaches 0 you will die
            cout    << "********************" << endLine
                    << "** YOU HAVE DIED!! **" << endLine
                    << "********************" << endLine;
            break;
        }

        if (currentRoom->treasureRoom) { // upon reaching the treasure room you have won the game
            cout    << "*******************************************************" << endLine
                    << "** Congratulations you have found the family treasue **" << endLine
                    << "*******************************************************" << endLine;
            break;
        }

        string action;

        cout << "What would you like to do?" << endLine;
        cout << "--------------------------" << endLine;
        cin >> action;
        cout << "--------------------------" << endLine;


        if (action == "help") { // after inputting 'help' a list of actions which can be used in the game will be listed
            cout << "Actions are
```

```
                'north','south','east','west','pickup','drop','use','quit'" << endLine;
            continue;
        }

        // exit game
        if (action == "quit") { // action which quits the game
            cout << "Thank you for playing... See you in game soon." << endLine;
            break;
        }

        // Movement options
        if (action == "north") { // action moves the player north of current room, if the player cannot go north an output will display and the player can change their action
            if (!currentPlayer.moveNorth(currentRoom->northWall->doorType))
                cout << ">>> Ouch! you are dazed and confused after hitting your head." << endLine;

            continue;
        }

        if (action == "south") { // action moves the player south of current room, if the player cannot go north an output will display and the player can change their action
            if (!currentPlayer.moveSouth(currentRoom->southWall->doorType))
                cout << ">>> Ouch! you are dazed and confused after hitting your head." << endLine;

            continue;
        }

        if (action == "east") { // action moves the player east of current room, if the player cannot go north an output will display and the player can change their action
            if (!currentPlayer.moveEast(currentRoom->eastWall->doorType))
                cout << ">>> Ouch! you are dazed and confused after hitting your head." << endLine;

            continue;
        }

        if (action == "west") { // action moves the player west of current room, if the player cannot go north an output will display and the player can change their action
            if (!currentPlayer.moveWest(currentRoom->westWall->doorType))
                cout << ">>> Ouch! you are dazed and confused after hitting your head." << endLine;

            continue;
        }

```

```cpp
        if (action == "pickup") { // action which picks up item from room inventory and puts it in player inventory
            int pickSlot;

            cout << "What do you want to pick up [Number]?" + endLine;
            cin >> pickSlot;

            if (pickSlot > -1
                && pickSlot < roomInventorySize) {
                inventory item = currentRoom->slots[pickSlot];

                if (item != inventory::emptyItem) {
                    if (currentPlayer.addInventory(item)) {
                        currentRoom->removeInventory(item);
                        continue;
                    }
                }
            }

            cout << ">>> Your bags are full or I can't find that item." << endLine; // if there is no room in player inventory, this message will display
            continue;
        }

        if (action == "drop") { // action which drops item from player inventory to room inventory
            int dropSlot;

            cout << "What do you want to drop [Number]?" + endLine;
            cin >> dropSlot;

            if (dropSlot > -1
                && dropSlot < playerInventorySize) {
                inventory item = currentPlayer.slots[dropSlot];

                if (item != inventory::emptyItem) {
                    if (currentPlayer.removeInventory(item)) {
                        currentRoom->addInventory(item);
                        continue;
                    }
                }
            }

            cout << ">>> Your bags are full or I can't find that item." << endLine; // if the player doesn't have the item typed, this message will display
            continue;
        }

        if (action == "use") { // action will use an item in the players inventory
```

```cpp
            int useSlot;

            cout << "What do you want to use [Number]?" + endLine;
            cin >> useSlot;

            if (useSlot > -1
                && useSlot < playerInventorySize) {
                inventory item = currentPlayer.slots[useSlot];

                if (item == inventory::matches) {     // if the item is matches and there is a lantern in player inventory, it will light up the lantern
                    for (int i = 0; i < playerInventorySize; i++) {
                        if (currentPlayer.slots[i] == inventory::lantern) {
                            currentPlayer.slots[i] = inventory::litLantern;
                            continue;
                        }
                    }
                }

                if (item == inventory::food) { // if the item is food and the player doesn't have max energy, the food will be used up from inventory and energy will increase by 5
                    if (!currentPlayer.eatFoods())
                        cout << ">>> You are full and cannot eat any more." << endLine;
                    else {
                        currentPlayer.removeInventory(inventory::food);
                        cout << "Yummy that was delicious." << endLine;
                    }

                    continue;
                }

                if (currentRoom->enableUnlock == item) { // if the item is a key it will unlock a locked door, if it is the wrong key nothing will happen
                    if (!currentRoom->unlockDoor())
                        cout << "Click... Nothing happened." << endLine;
                    else
                        cout << "Click... You've unlocked the door." << endLine;

                    continue;
                }
            }

            cout << ">>> You cannot do that right now!" << endLine; // if the action can't be done this message will display
            continue;
        }

        cout << ">>> I'm sorry I don't understand what you mean! (help
            available)" << endLine; // if a command which is unknown is input, this
             message will display
    };

    return 0;
}
```