|  |
| --- |
| #include <stdlib.h> |
|  | #include <stdio.h> |
|  | #include <stdbool.h> |
|  | #include <string.h> |
|  |  |
|  | #define STATUS\_SUCCESS 0 |
|  | #define STATUS\_ERR 1 |
|  |  |
|  | #define STATE\_IN\_BOARD 1 |
|  | #define STATE\_OUT\_OF\_BOARD 2 |
|  | #define STATE\_EDGE\_OF\_BOARD 4 |
|  | #define STATE\_ARRIVED\_BEFORE 8 |
|  |  |
|  | #define STATE\_IS(state, mask) (state & mask) |
|  | #define ARRAY\_GET(array, col\_count, row, col) (array[col\_count \* (row - 1) + (col - 1)]) |
|  | #define ARRAY\_GET\_POINTER(array, col\_count, row, col) (&(array[col\_count \* (row - 1) + (col - 1)])) |
|  |  |
|  | typedef struct Pos { |
|  | int x; |
|  | int y; |
|  | } Pos, \*pPos; |
|  |  |
|  | int checkHorseState(pPos cboardSet, pPos pHorse) { |
|  | if ( |
|  | pHorse->x < 1 || pHorse->y < 1 || |
|  | pHorse->x > cboardSet->x || pHorse->y > cboardSet->y |
|  | ) { |
|  | return STATE\_OUT\_OF\_BOARD; |
|  | } |
|  |  |
|  | if ( |
|  | pHorse->x == 1 || pHorse->x == cboardSet->x || |
|  | pHorse->y == 1 || pHorse->y == cboardSet->y |
|  | ) { |
|  | return STATE\_EDGE\_OF\_BOARD | STATE\_IN\_BOARD; |
|  | } |
|  |  |
|  | return STATE\_IN\_BOARD; |
|  | } |
|  |  |
|  | int printChessBoard(pPos cboardSet, pPos pHorse) { |
|  | int state = checkHorseState(cboardSet, pHorse); |
|  | if (state == STATE\_OUT\_OF\_BOARD) { |
|  | return STATUS\_ERR; |
|  | } |
|  |  |
|  | for (unsigned int i = cboardSet->y; i >= 1; i--) { |
|  | printf\_s(" %u ", i); |
|  | for (unsigned int j = 1; j <= cboardSet->x; j++) { |
|  | if (i == pHorse->y && j == pHorse->x) { |
|  | printf\_s(" A "); |
|  | } |
|  | else { |
|  | printf\_s(" . "); |
|  | } |
|  | } |
|  | printf\_s("\r\n"); |
|  | } |
|  | printf\_s("y/x"); |
|  | for (unsigned int i = cboardSet->x; i >= 1; i--) { |
|  | printf\_s(" %u ", cboardSet->x - i + 1); |
|  | } |
|  | printf\_s("\r\n"); |
|  |  |
|  | return STATUS\_SUCCESS; |
|  | } |
|  |  |
|  | int printChessStep(pPos cboardSet, int \*chessStep) { |
|  | for (unsigned int i = cboardSet->y; i >= 1; i--) { |
|  | printf\_s(" %3u ", i); |
|  | for (unsigned int j = 1; j <= cboardSet->x; j++) { |
|  | if (ARRAY\_GET(chessStep, cboardSet->x, i, j)) { |
|  | printf\_s(" %3d ", ARRAY\_GET(chessStep, cboardSet->x, i, j)); |
|  | } |
|  | else { |
|  | printf\_s(" . "); |
|  | } |
|  | } |
|  | printf\_s("\r\n"); |
|  | } |
|  | printf\_s(" y/x "); |
|  | for (unsigned int i = cboardSet->x; i >= 1; i--) { |
|  | printf\_s(" %3u ", cboardSet->x - i + 1); |
|  | } |
|  | printf\_s("\r\n"); |
|  |  |
|  | return STATUS\_SUCCESS; |
|  | } |
|  |  |
|  | int moveChess(pPos cboardSet, pPos pHorseInit, pPos pHorseCurrnet, int \*chessStep, bool edgeReached, int step) { |
|  | if (ARRAY\_GET(chessStep, cboardSet->x, pHorseCurrnet->y, pHorseCurrnet->x) > 1) { |
|  | return STATE\_ARRIVED\_BEFORE; |
|  | } |
|  |  |
|  | if (STATE\_IS(checkHorseState(cboardSet, pHorseCurrnet), STATE\_OUT\_OF\_BOARD)) { |
|  | return STATE\_OUT\_OF\_BOARD; |
|  | } |
|  |  |
|  | if (STATE\_IS(checkHorseState(cboardSet, pHorseCurrnet), STATE\_EDGE\_OF\_BOARD)) { |
|  | edgeReached = true; |
|  | } |
|  |  |
|  | int \*chessStepOri = chessStep; |
|  | chessStep = malloc(cboardSet->x \* cboardSet->y \* sizeof(int)); |
|  | memcpy(chessStep, chessStepOri, cboardSet->x \* cboardSet->y \* sizeof(int)); |
|  | \*(ARRAY\_GET\_POINTER(chessStep, cboardSet->x, pHorseCurrnet->y, pHorseCurrnet->x)) = step; |
|  |  |
|  | if ( |
|  | pHorseCurrnet->x == pHorseInit->x && |
|  | pHorseCurrnet->y == pHorseInit->y && |
|  | edgeReached |
|  | ) { |
|  | \*(ARRAY\_GET\_POINTER(chessStep, cboardSet->x, pHorseCurrnet->y, pHorseCurrnet->x)) = 1; |
|  | printf\_s("ok:\r\n"); |
|  | printChessStep(cboardSet, chessStep); |
|  | printf\_s("\r\n"); |
|  | return STATUS\_SUCCESS; |
|  | } |
|  |  |
|  | Pos newHorseCurrent; |
|  | newHorseCurrent.x = pHorseCurrnet->x + 1; |
|  | newHorseCurrent.y = pHorseCurrnet->y + 2; |
|  | moveChess(cboardSet, pHorseInit, &newHorseCurrent, chessStep, edgeReached, step + 1); |
|  |  |
|  | newHorseCurrent.x = pHorseCurrnet->x - 1; |
|  | newHorseCurrent.y = pHorseCurrnet->y + 2; |
|  | moveChess(cboardSet, pHorseInit, &newHorseCurrent, chessStep, edgeReached, step + 1); |
|  |  |
|  | newHorseCurrent.x = pHorseCurrnet->x - 1; |
|  | newHorseCurrent.y = pHorseCurrnet->y - 2; |
|  | moveChess(cboardSet, pHorseInit, &newHorseCurrent, chessStep, edgeReached, step + 1); |
|  |  |
|  | newHorseCurrent.x = pHorseCurrnet->x + 1; |
|  | newHorseCurrent.y = pHorseCurrnet->y - 2; |
|  | moveChess(cboardSet, pHorseInit, &newHorseCurrent, chessStep, edgeReached, step + 1); |
|  |  |
|  | newHorseCurrent.x = pHorseCurrnet->x + 2; |
|  | newHorseCurrent.y = pHorseCurrnet->y + 1; |
|  | moveChess(cboardSet, pHorseInit, &newHorseCurrent, chessStep, edgeReached, step + 1); |
|  |  |
|  | newHorseCurrent.x = pHorseCurrnet->x - 2; |
|  | newHorseCurrent.y = pHorseCurrnet->y + 1; |
|  | moveChess(cboardSet, pHorseInit, &newHorseCurrent, chessStep, edgeReached, step + 1); |
|  |  |
|  | newHorseCurrent.x = pHorseCurrnet->x - 2; |
|  | newHorseCurrent.y = pHorseCurrnet->y - 1; |
|  | moveChess(cboardSet, pHorseInit, &newHorseCurrent, chessStep, edgeReached, step + 1); |
|  |  |
|  | newHorseCurrent.x = pHorseCurrnet->x + 2; |
|  | newHorseCurrent.y = pHorseCurrnet->y - 1; |
|  | moveChess(cboardSet, pHorseInit, &newHorseCurrent, chessStep, edgeReached, step + 1); |
|  |  |
|  | free(chessStep); |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) { |
|  | Pos cboardSet = { 0 }; |
|  | Pos horsePos = { 0 }; |
|  | int \*chessStep; |
|  |  |
|  | printf\_s("Chessboard Size: "); |
|  | scanf\_s("%d%d", &cboardSet.x, &cboardSet.y); |
|  |  |
|  | printf\_s("Horse Position: "); |
|  | scanf\_s("%d%d", &horsePos.x, &horsePos.y); |
|  |  |
|  | printChessBoard(&cboardSet, &horsePos); |
|  |  |
|  | chessStep = malloc(cboardSet.x \* cboardSet.y \* sizeof(int)); |
|  | memset(chessStep, 0, cboardSet.x \* cboardSet.y \* sizeof(int)); |
|  |  |
|  | moveChess(&cboardSet, &horsePos, &horsePos, chessStep, false, 1); |
|  | free(chessStep); |
|  |  |
|  | getchar(); |
|  | getchar(); |
|  |  |
|  | return STATUS\_SUCCESS; |
|  | } |