5/1/2016

Alfredo Diaz

CSC 17A 42824

Project 1

Minesweeper

Introduction

For our project I chose to write up a program that allows the user to play Minesweeper. The game is played in the command prompt using the standard row and column grid or a an “X’ and “Y” coordinate plane. The game randomly places mines on the game board that the user will define before playing. The goal is simple, navigate around the board without revealing mines as you go, and you accomplish this by using clues given about surrounding mines from your previous choice. This is done when you select one potential clear zone it will denote a number that will tells you how many mines are located from that position selected. The numbers can range from zero to eight.

This project is considered important because it allowed for a smoother transition in to utilizing classes. This is due to the fact that I was able utilize some of the same declaration and using a structure. By learning a few of the syntax for structure manipulation it allowed for me to prepare for some of the similar syntax used by classes, thus allowing for me to come into classes a bit more prepared.

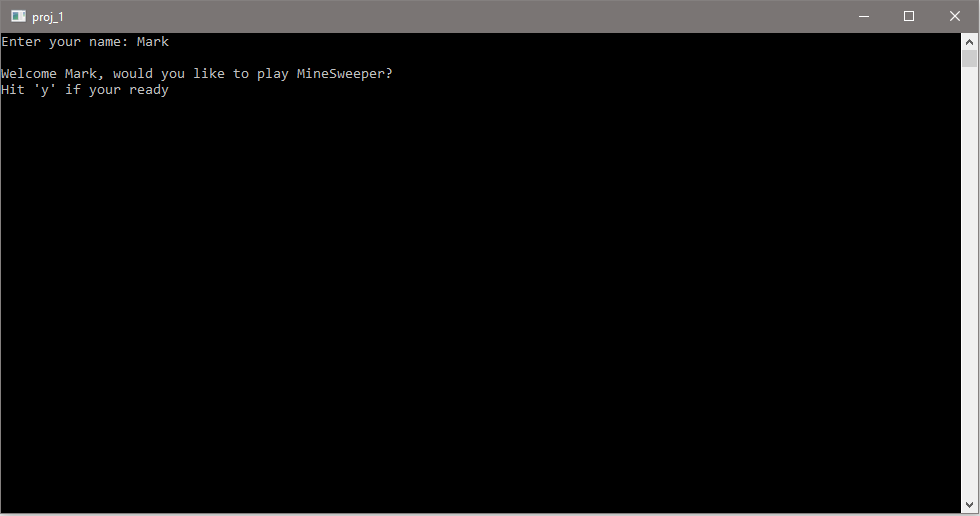
Summary

This project came in about a total length of 489 lines excluding blank lines and comments, and 599 in total. In the program I utilized dynamic memory allocation when asking for the user’s name and for the playing field. In addition, functions with structures were used with the creation of the playing field and many other areas. Pointers were used for the name and for passing information to functions. Character and string arrays were used. Reading and writing to a binary file was a bit more difficult and is the area that I am not a hundred percent as I receive an action, but I am not sure if that is what I truly wanted. The project was not to difficult just time consuming I believe I spend about three days cumulative over the course of a few weeks alongside my regular course work.

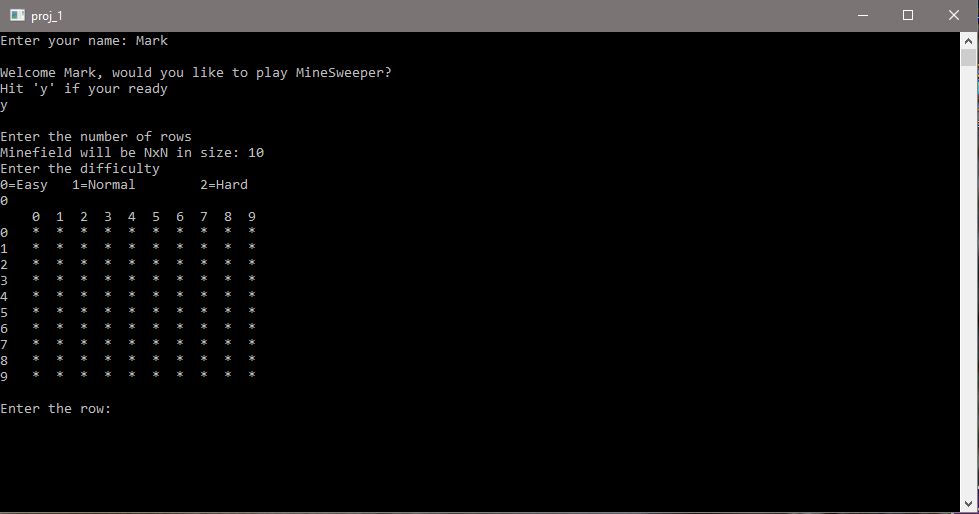
Description

The project was written to be built dynamically as the user provides data that was asked of them i.e. name, rows in order to construct the grid in a row times row format, and their selection for the following picture the descriptions will come before each instance.

For example, if you provide It first with any name such as Mark it will retain that name in order to congratulate you, or comfort you depending if you win or lose the round.

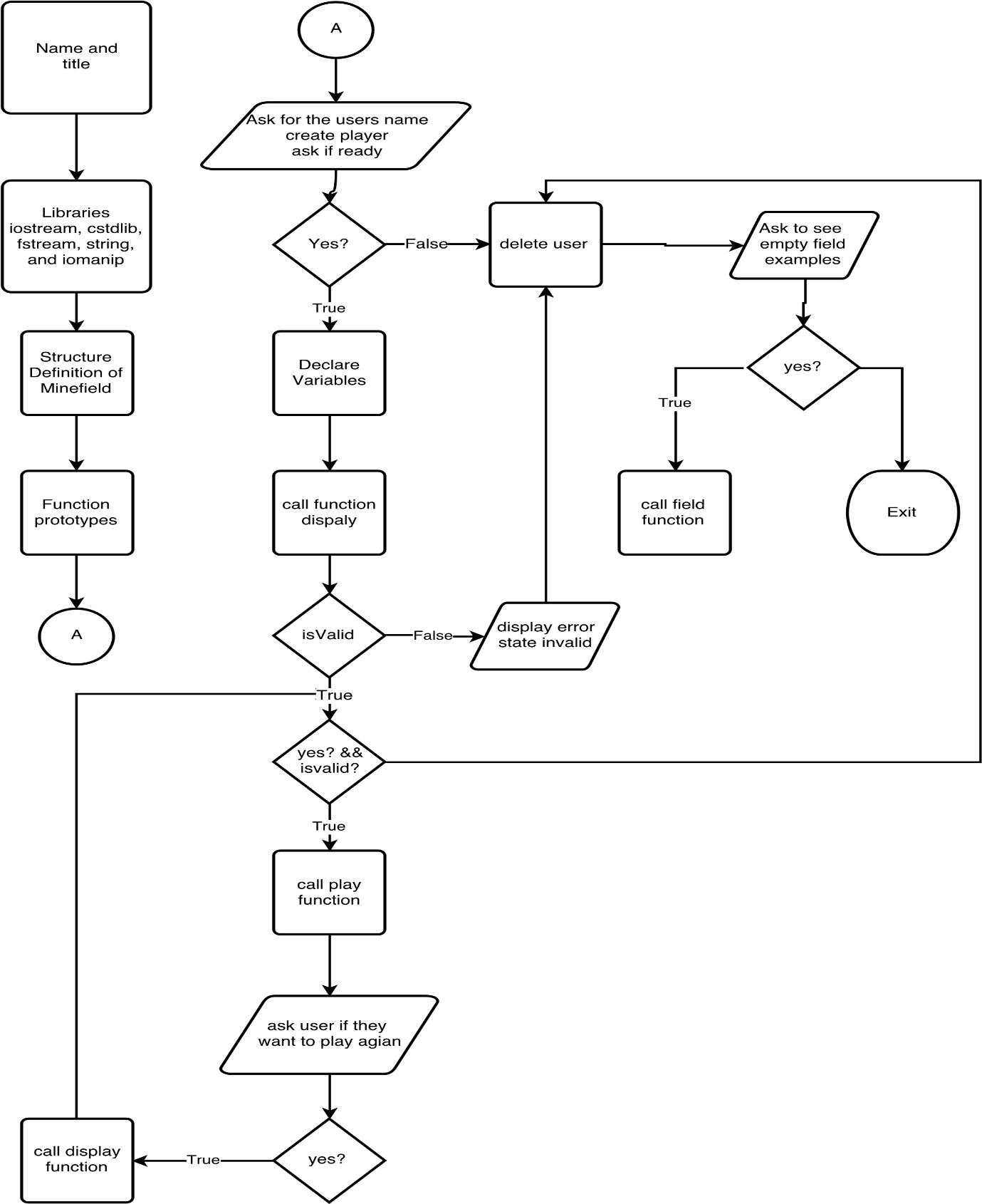


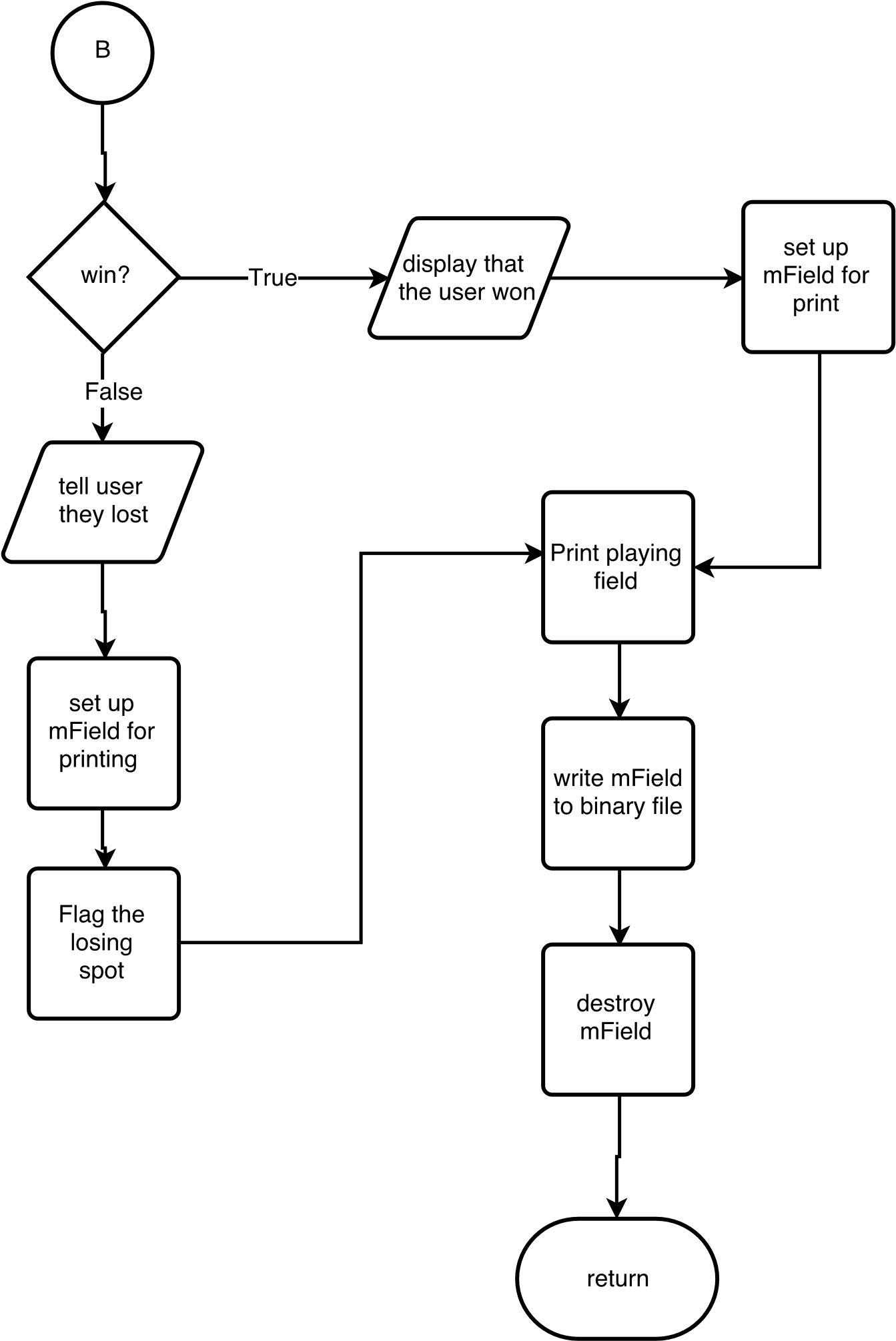
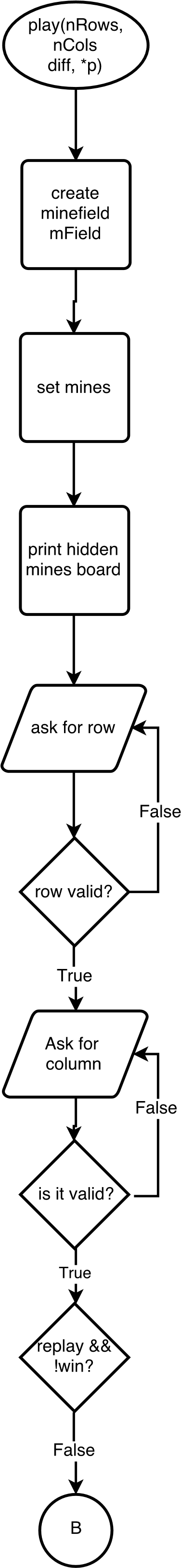
Then it will ask for you to input your choice if you would like to play or exit the program. Followed by asking for a valid size for the array of units, and prevents you from continuing on i.e. a ten by ten grid and for the difficulty which determines how many mine will be placed on the board.



Furthermore. It will ask you for your choice in where you want to reveal followed by displaying an updated game board. For the picture I used I lost, I’m not a patient man when I comes to playing minesweeper I prefer the speed random selection method, but in doing so it demonstrates that it will reveal the board and shows you your last move made by flagging it as a “T”.



Flow Chart



Pseudo Code

1. Ask user for their name.
2. Create the player using their name.
3. Ask if the user is ready to play the game.
4. If they choose yes.
   1. Begin playing the game.
   2. Continue until it is calculated that the player has won or lost.
   3. If the game is over, ask if they want to play again.
      1. If yes re run steps I to IV c.
      2. If no exit game.
5. If no delete the user and exit the game.
6. Ask if you want to see previous results of last game.
   1. If yes read from binary file.
   2. If no continue to step VII.
7. Ask if they want to see examples of random mine field.
8. If no, then exit program
   1. Else display fields.

Major Variables

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Variable Name | Description | Location |
| short | \*\*board | Holds the game board data. | Line 36 in MineField structure. |
| short | row | Holds the number of rows. | Line 38 in MineField structure |
| short | column | Hold the number of columns. | Line 40 in MineField structure |
| short | mines | Holds the mines data | Line 42 in MineField structure |
| enum | flags | Holds the status condition of the spaces on the field. | Line 34 in MineField structure |
| enum | difficulty | Determines how many mines to set. | Line 32 in MineField structure |
| char\* | name | Holds the name of the user. | Line 74 in main () |

Concept From Chapter’s

|  |  |  |  |
| --- | --- | --- | --- |
| Chapter | Concept | Description | Location |
| 9.2 | Pointer Variables | Used to hold memory addresses, and allows for indirect data manipulation. | Lines 64, 74, 161, 162, 205, 216, 222, 224, 261, 271…etc. |
| 9.7 | Passing pointers to functions | Passing a pointer gives the function access to the original argument | Lines 64 and 66 (functions).  Lines 158 - 201 (play()) and 205 - 218(\*name()) |
| 10 | Working with char | Using and manipulation of char arrays | Lines 205 - 218 |
| 11 | structures | ADT’s created to have their own domain of data and accessing structures to create objects. | Linea 30 – 42 Minefield struct.  Lines 86, 159 – 163, 222 – 239. |
| 12 | Writing and reading to binary file | Use fstream library to write/read data to file | Lines 573 - 598 |

Reference

I used a video on YouTube to get me started:

Writing MineSweeper in 10 minutes

By Anton Te

<https://www.youtube.com/watch?v=vqJQoangCSw>

Code

/\*

\*

\* Author: Diaz, Alfredo - Project 1

\* Project Minesweeper

\*

\*/

///System Libraries

#include <cstdlib>

#include <iostream>

#include <fstream>

#include <string>

#include <iomanip>

using namespace std;

///User Libraries

///Global Constants

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Structure

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// Structure that holds the minefield, and

/// the associated flags that occur when

/// the user selects a square.

struct MineField {

///Difficulty

enum Difficulty {EASY, NORMAL, HARD};

///Flags for various possibilities

enum flags {EMPTY=10, MINE, CLEAR, LOSS};

///Game board

short \*\*board;

///Number of rows on board

short row;

///Number of columns on the board

short column;

///Number of mines on the playing field

short mines;

};

///Function Prototypes

MineField \*create(short,short);

void destroyBoard(MineField \*);

void printClear(MineField \*);

void print(MineField \*);

MineField::Difficulty shortToDiff(short);

bool isValid(short, short, MineField::Difficulty);

bool isClear(MineField \*, short, short);

short numMines(MineField::Difficulty);

void placeMines(MineField \*);

void placeFlags(MineField \*);

short adjacent(MineField \*, short, short, short = MineField::MINE);

void clearArea(MineField \*, short, short);

void perimeter(MineField \*);

void zero(MineField \*, short, short);

bool win(MineField \*);

void field();

bool replay(MineField \*, short, short);

void play(short, short, MineField::Difficulty, char\*);

void display(short&, MineField::Difficulty&);

char \*name();

void writeBin(MineField \*, string);

void readBin(string);

///Execution Begins Here!

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

/// Obtain user's name

char \*user = name();

/// Determine if the player wishes to play a game

cout << "\nWelcome " << user << ", would you like to play MineSweeper?\n"

<< "Hit 'y' if your ready\n";

char choice;

cin >> choice;

if(choice == 'y'){

/// creation of game board

short nRows;

MineField::Difficulty difficult;

/// Set game parameters from user input

display(nRows, difficult);

/// determine validity

if(isValid(nRows, nRows, difficult)){

while(choice == 'y' && isValid(nRows, nRows, difficult)){

play(nRows, nRows, difficult, user);

cout << endl;

cout << user << ", Would you like to play agian?" << endl;

cout << "Input 'y' for yes or anything else for no\n";

cin >> choice;

cout << endl;

/// recreate board if yes

if(choice == 'y' ) display(nRows,difficult);

}

}

/// if data is not valid

else

cout << "Board too small.";

}

cout << "\nGame over.\n";

cout << "\nGoodbye, " << user << endl;

/// clean us used memory

delete user;

readBin("result");

cout << "Would you like to see some empty mine fields "

"stored in a structure?\n"

"Hit 'y' for yes: ";

cin >> choice;

if (choice == 'y'){

field();

}

cout << endl;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Function definitions

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display(short &rows, MineField::Difficulty &d) {

cout << "\nEnter the number of rows\n"

"Minefield will be NxN in size: ";

cin >> rows;

short diff;

cout << "Enter the difficulty\n"

"0=Easy\t 1=Normal\t 2=Hard\n";

cin >> diff;

d = shortToDiff(diff);

}

/// Function while return true if input chosen was valid

bool isValid(short rows, short cols, MineField::Difficulty diff) {

/// check to make sure number of mines does not exceed the number of spots

/// available

return (rows \* cols) > numMines(diff);

}

/// Play a game of minesweeper

/// User inputs how many rows and columns and the difficulty

void play(short nRows, short nCols,

MineField::Difficulty diff, char \*p) {

srand(static\_cast<unsigned int>(time(0)));

MineField \*mField = create(nRows, nCols);

MineField \*result;

mField->mines = numMines(diff);

placeMines(mField);

print(mField);

short row, col;

do {

/// Select the row

do {

cout << "Enter the row: ";

cin >> row;

/// check bounds

} while (row < 0 || row >= mField->row);

do {

cout << "Enter the column: ";

cin >> col;

/// check bounds

} while (col < 0 || col >= mField->column);

cout << endl;

} while (replay(mField, row, col) && !win(mField));

/// Prepare to print completed minefield

if (win(mField)) {

cout << p << ", You win\n";

placeFlags(mField);

}

else{

cout << p << ", you have lost\n";

placeFlags(mField);

mField->board[row][col]= MineField::LOSS;

}

/// Print the complete minefield

printClear(mField);

/// write result to binary file

writeBin(mField, "result");

/// deallocate the game area

destroyBoard(mField);

}

/// Function gets the user name as a string converts it to a char array

/// for the 1d dynamic array requirement

char \*name() {

cout << "Enter your name: ";

string in;

cin >> in;

short size = in.size();

/// make room for '\0'

char \*name = new char[size+1];

for (short i = 0; i != size; ++i) {

\*(name+i) = in[i];

}

\*(name+size+1) = '\0';

return name;

}

/// Function that creates the grid on which game will be played

MineField\* create(short rows, short cols) {

/// dynamically create a minefield

MineField \*out = new MineField;

out->row=rows;

out->column = cols;

/// Create the 2D game minefield

out->board = new short \*[rows];

/// Create each row

for (short row = 0; row != rows; ++row)

out->board[row] = new short [cols];

/// Make sure each square is empty

for (short i = 0; i != rows; ++i)

for (short j = 0; j != rows; ++j)

out->board[i][j] = MineField::EMPTY;

return out;

}

/// Function return the MineField::Difficulty type from

/// the short variable

MineField::Difficulty shortToDiff(short choice) {

switch (choice) {

case (0):

return MineField::Difficulty::EASY;

break;

case (1):

return MineField::Difficulty::NORMAL;

break;

case (2):

return MineField::Difficulty::HARD;

default:

return MineField::Difficulty::EASY;

break;

}

}

/// Function deallocates memory

void destroyBoard(MineField \*mField) {

/// delete each dynamically allocated row

for (short i = 0; i != mField->row; ++i)

delete[] mField->board[i];

/// delete the dynamically allocated structure

delete mField;

}

/// Functions prints the minefield with all the squares revealed.

/// used mostly after player loses

void printClear(MineField\* mField) {

for (short row = 0; row != mField->row; ++row){

for (short col = 0; col != mField->column; ++col) {

if ( \*(\*(mField->board+row) + col) == MineField::LOSS)

cout << "T ";

else if (\*(\*(mField->board+row) + col) == MineField::MINE)

cout << "x ";

else if (!isClear(mField, row, col))

cout << adjacent(mField, row, col) << " ";

else

cout << "0 ";

}

cout << endl;

}

cout << endl;

}

/// Function prints the minefield with spaces hidden

void print(MineField\* mField) {

/// Print the column index

for (short i = 0; i != mField->column; ++i){

/// Pad initial output of column indicator

if (i==0)

cout << " ";

cout << setw(3) << i;

}

cout << endl;

for (short row = 0; row != mField->row; ++row){

for (short col = 0; col != mField->column; ++col){

if(col == 0 && row < 10) cout << row << " ";

if (col == 0 && row >= 10) cout << row << " ";

/// KEEP EMPTY spaces and MINEs hidden

if (mField->board[row][col] == MineField::EMPTY ||

mField->board[row][col] == MineField::MINE)

cout << setw(3) << right << "\* ";

/// print out the CLEARed area

else if (mField->board[row][col] == MineField::CLEAR)

cout << setw(2)<< 0 << " ";

/// Print out the actual value of the square

else

cout << setw(2)<< mField->board[row][col] << " ";

}

cout << endl;

}

cout << endl;

}

/// Function returns the number of mines to set based on Difficulty

short numMines(MineField::Difficulty d) {

if (d==MineField::EASY)

return 15;

else if (d==MineField::NORMAL)

return 30;

else

return 45;

}

/// Function places mines in grid

void placeMines(MineField \*mField) {

/// holds how many mines will be used

short mines = mField->mines;

/// keep looping through minefield until all mines are set

while (mines) {

for (short i = 0; i != mField->row; ++i) {

for (short j = 0; j != mField->column; ++j) {

/// place mines if result of rand()%15 == 0

if ((rand() % 100) % 10 == 0){

///only place mines if mines are still available

/// and current is empty

if (mines && mField->board[i][j] == MineField::EMPTY) {

/// set the mine

mField->board[i][j] = MineField::MINE;

--mines;

}

}

}

}

}

}

/// Function returns how many 'flag' elements surround a given square

short adjacent(MineField \*mField, short row, short col, short FLAG) {

short nAd=0; /// the number of adjacent mines

/// not on first or last row or first or last column

/// most of the searches take place in this area

if ( row > 0 && col > 0 && row < mField->row-1 && col < mField->column-1) {

/// search the 3x3 grid surrounding a cell

for (short i = row-1; i <= row+1; ++i) {

for (short j = col-1; j <= col+1; ++j)

if (mField->board[i][j] == FLAG)

++nAd;

}

}

/// on the first row, not on first or last column

else if ( row == 0 && col > 0 && col < mField->column - 1) {

for (short i = row; i <= row+1; ++i) {

for (short j = col-1; j <= col+1; ++j)

if (mField->board[i][j] == MineField::MINE)

++nAd;

}

}

/// on the last row, not on first or last column

else if ( row == mField->row-1 && col > 0 && col < mField->column - 1) {

for (short i = row-1; i <= row; ++i) {

for (short j = col-1; j <= col+1; ++j)

if (mField->board[i][j] == MineField::MINE)

++nAd;

}

}

/// on the first column, not on first or last row

/// search to the right

else if ( col == 0 && row > 0 && row < mField->row - 1) {

for (short i = row-1; i <= row+1; ++i) {

for (short j = col; j <= col+1; ++j)

if (mField->board[i][j] == MineField::MINE)

++nAd;

}

}

/// on the last column, not on first or last row

/// search to the left

else if ( col == mField->column-1 && row > 0 && row < mField->row - 1) {

for (short i = row-1; i <= row+1; ++i) {

for (short j = col-1; j <= col; ++j)

if (mField->board[i][j] == MineField::MINE)

++nAd;

}

}

/// top left corner

else if (row == 0 && col == 0) {

if (mField->board[row][col+1] == MineField::MINE) ++nAd;

if (mField->board[row+1][col] == MineField::MINE) ++nAd;

if (mField->board[row+1][col+1] == MineField::MINE) ++nAd;

}

/// top right corner

else if (row == 0 && col == mField->column-1) {

if (mField->board[row][col-1] == MineField::MINE) ++nAd;

if (mField->board[row+1][col] == MineField::MINE) ++nAd;

if (mField->board[row+1][col-1] == MineField::MINE) ++nAd;

}

/// bottom left corner

else if (row == mField->row-1 && col == 0) {

if (mField->board[row-1][col] == MineField::MINE) ++nAd;

if (mField->board[row-1][col+1] == MineField::MINE) ++nAd;

if (mField->board[row][col+1] == MineField::MINE) ++nAd;

}

/// bottom right corner

else if (row == mField->row-1 && col == mField->column-1) {

if (mField->board[row-1][col-1] == MineField::MINE) ++nAd;

if (mField->board[row-1][col] == MineField::MINE) ++nAd;

if (mField->board[row][col-1] == MineField::MINE) ++nAd;

}

/// return number of mines from appropriate if statement

return nAd;

}

/// Function is true if there 0 land mines adjacent to

/// selected square

bool isClear(MineField \* mField, short row, short col) {

if (adjacent(mField, row, col))

return false; /// there was at least one mine adjacent

return true; /// area was clear

}

/// Clear an area whose values are clear

/// i.e 0 adjacent mines

void zero(MineField \*mField, short row, short col) {

/// check bounds

if ( row >= mField->row || row < 0 || col >= mField->column || col < 0)

return;

if (isClear(mField, row, col) && mField->board[row][col] != MineField::CLEAR){

mField->board[row][col] = MineField::CLEAR;

/// go up one row

zero(mField, row+1, col);

/// go down one row

zero(mField, row-1, col);

/// go right one col

zero(mField, row, col+1);

/// go left one col

zero(mField, row, col-1);

}

/// space was not clear or already shown

else

return;

}

/// Function shows how many mines are adjacent to selected square

/// for the entire minefield

void placeFlags(MineField \*mField) {

for (short i = 0; i != mField->row; ++i)

for (short j = 0; j != mField->column; ++j)

/// don't look for adjacent mines in areas where

/// mine is already located

if (mField->board[i][j] != MineField::MINE)

mField->board[i][j] = adjacent(mField, i, j);

}

/// Function reveals what is underneath the square that the user has selected

/// and whether to continue based on what is revealed

/// i.e selecting a mine means you lost, game over

bool replay(MineField \* mField, short row, short col) {

/// check if user selected a losing square

if (mField->board[row][col] == MineField::MINE)

return false;

/// Square is a zero, clear the surrounding area if necessary

else if (isClear(mField, row, col) ){

zero(mField, row, col); /// show cleared area

perimeter(mField);

print(mField);

return true;

}

/// Square had adjacent mine

/// reveal the number to the user

else {

mField->board[row][col] = adjacent(mField, row, col);

print(mField);

return true;

}

}

/// Function checks whether the player has won

bool win(MineField \*mField) {

for (short i = 0; i != mField->row; ++i)

for (short j = 0; j != mField->column; ++j)

/// if there are empty spaces player has not won

if (mField->board[i][j] == MineField::EMPTY)

return false;

/// there were no empty spaces left. Player has won

return true;

}

/// Function find the perimeter of the cleared areas

void perimeter(MineField \*mField) {

for (short row = 0; row != mField->row; ++row ) {

/// avoid search at left and right edge of array

for (short col = 0; col != mField->column; ++col) {

/// when you're not on the bounds of the array

if (row > 0 && row < mField->row-1

&& col > 0 && col <mField->column-1)

if (mField->board[row][col] == MineField::CLEAR) {

/// check that the previous number has mines adjacent

if (mField->board[row][col-1] != MineField::CLEAR)

mField->board[row][col-1] = adjacent(mField, row, col-1);

/// check if the next number has mines adjacent

if (mField->board[row][col+1] != MineField::CLEAR)

mField->board[row][col+1] = adjacent(mField,row, col+1);

if (mField->board[row-1][col] != MineField::CLEAR)

mField->board[row-1][col] = adjacent(mField, row-1, col);

/// check if the next number has mines adjacent

if (mField->board[row+1][col] != MineField::CLEAR)

mField->board[row+1][col] = adjacent(mField,row+1, col);

/// check the adjacent corners

if (mField->board[row+1][col-1] != MineField::CLEAR)

mField->board[row-1][col-1] = adjacent(mField,row-1, col-1);

if (mField->board[row-1][col+1] != MineField::CLEAR)

mField->board[row-1][col+1] = adjacent(mField,row-1, col+1);

if (mField->board[row+1][col-1] != MineField::CLEAR)

mField->board[row+1][col-1] = adjacent(mField,row+1, col-1);

if (mField->board[row+1][col+1] != MineField::CLEAR)

mField->board[row+1][col+1] = adjacent(mField,row+1, col+1);

}

}

}

}

/// This function creates an array of the Minefield structure

/// as part of the requirement's to be able to write to and read

/// from an array of structures

void field() {

cout << "How many mine fields do you want to see: ";

int n;

cin >> n;

MineField \*\*mField = new MineField\*[n];

const int row = 10;

const int col = 10;

/// create the fields

for (int i = 0; i != n; ++i) {

/// Create each field

mField[i] = create(row, col);

/// get number of mines

mField[i]->mines = numMines(MineField::EASY);

/// set the mines

placeMines(\*(mField+i));

/// set the flags

placeFlags(\*(mField+i));

/// print the field

printClear(\*(mField+i));

cout << endl;

}

cout << endl;

/// deallocate memory

for (int i = 0; i != n; ++i) {

destroyBoard(\*(mField+i));

}

delete []mField;

}

/// Function writes the minefield structure to a binary file

void writeBin(MineField \*mField, string fileName) {

/// Write the result to a binary file

fstream out(fileName.c\_str(), ios::out | ios::binary); /// open the file

out.write(reinterpret\_cast<char \*>(&mField),sizeof(\*mField)); /// write to the file

out.close();

}

/// Function prints the data variable from the Minefield structure

/// written to a binary file

void readBin(string fileName) {

/// Ask user if they want to see the result of the last game

char response;

cout << "Would you like to see the result of the last game as "

"read from a binary file?\n"

"Hit 'y' if yes: ";

cin >> response;

if (response == 'y') {

cout << "\nResult of your last game:\n";

/// Create space to hold the file read

MineField \*result;

fstream in(fileName.c\_str(), ios::in | ios::binary);

in.read(reinterpret\_cast<char \*>(&result), sizeof(\*result));

printClear(result);

in.close();

}

}