P2 Latin Squares

Start Assignment

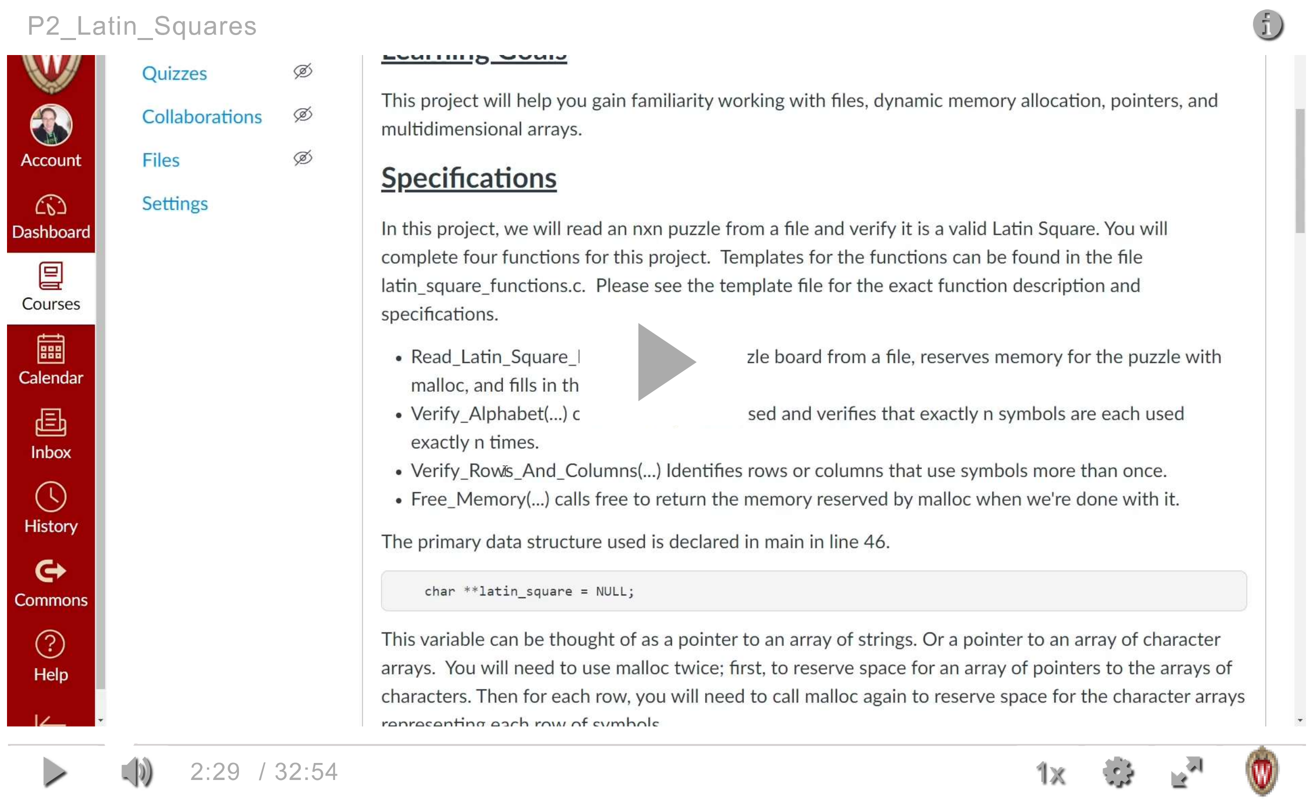
Due Sunday by 11:59pm **Points** 50 **Submitting** a file upload **File Types** c **Available** Jul 7 at 12am - Aug 9 at 11:59pm about 1 month

Project 2: Latin Squares

A Latin Square is an n x n puzzle filled with n different symbols. Each symbol occurs exactly once in each row and exactly once in each column. The following three puzzles are all examples of valid Latin Squares.

```
abcd        123        %#
bcda        312        #%
cdab        231
dabc
```

Video Intro



This project will help you gain familiarity working with files, dynamic memory allocation, pointers, and multidimensional arrays.

Specifications

In this project, we will read an nxn puzzle from a file and verify it is a valid Latin Square. You will complete four functions for this project. Templates for the functions can be found in the file latin_square_functions.c. Please see the template file for the exact function description and specifications.

- Read_Latin_Square_I zle board from a file, reserves memory for the puzzle with malloc, and fills in th
- Verify_Alphabet(...) c sed and verifies that exactly n symbols are each used exactly n times.
- Verify_Rows_And_Columns(...) Identifies rows or columns that use symbols more than once.
- Free_Memory(...) calls free to return the memory reserved by malloc when we're done with it.

The primary data structure used is declared in main in line 46.

```
char **latin_square = NULL;
```

This variable can be thought of as a pointer to an array of strings. Or a pointer to an array of character arrays. You will need to use malloc twice; first, to reserve space for an array of pointers to the arrays of characters. Then for each row, you will need to call malloc again to reserve space for the character arrays

Corrections and Additions

1. Do not use any square brackets in your work. Not even to create arrays. Instead, just call malloc for all array creation.
2. The largest valid puzzle uses all 94 valid symbols, so the largest valid filesize is 8931 bytes. Invalid puzzles may use duplicate symbols and be even larger (without limit). We will not test puzzles larger than 100x100.
3. Only one error message is printed per row/column even if more than one symbol is used multiple times in a single row.

Learning Goals

This project will help you gain familiarity working with files, dynamic memory allocation, pointers, and multidimensional arrays.

Specifications

In this project, we will read an nxn puzzle from a file and verify it is a valid Latin Square. You will complete four functions for this project. Templates for the functions can be found in the file latin_square_functions.c. Please see the template file for the exact function description and specifications.

- Read_Latin_Square_File(...) reads the puzzle board from a file, reserves memory for the puzzle with malloc, and fills in the data structure.
- Verify_Alphabet(...) counts the symbols used and verifies that exactly n symbols are each used exactly n times.
- Verify_Rows_And_Columns(...) Identifies rows or columns that use symbols more than once.
- Free_Memory(...) calls free to return the memory reserved by malloc when we're done with it.

The primary data structure used is declared in main in line 46.

```
    char **latin_square = NULL;
```

This variable can be thought of as a pointer to an array of strings. Or a pointer to an array of character arrays. You will need to use malloc twice; first, to reserve space for an array of pointers to the arrays of characters. Then for each row, you will need to call malloc again to reserve space for the character arrays representing each row of symbols.

All puzzles tested will be square. They will have the same number of rows as columns.

Valid symbols include any character with **ascii code** (https://man7.org/linux/man-pages/man7/ascii.7.html) 33 to 126 inclusive. The files will not use invalid characters - they will have new lines and end of file symbols.

All work must be done in the latin_square_functions.c file. Do not turn in the latin_square_main.c file. We will not look at it.

No [ ] Allowed Your Functions

A key objective of this assignment is for you to practice using pointers. To achieve this, you are **not allowed** to use array indexing with square brackets to access arrays in the latin_square_functions.c file. Instead, you are required to use address arithmetic and dereferencing to access arrays. You may not use brackets, e.g., writing “latin_square[1][2]” to access square (1,2) of the game board. Submitting a solution using indexing to access the puzzle or any other array will result in a **50% reduction of your score**.

Files

- **latin_square_main.c** (https://canvas.wisc.edu/courses/248855/files/20656958/download?download_frd=1)   // your code should work with the original latin_square_main.c file
- **latin_square_functions.c** (https://canvas.wisc.edu/courses/248855/files/20656957/download?download_frd=1)   // do your work here
- Test Files
  - **Latin_4x4.txt** (https://canvas.wisc.edu/courses/248855/files/20656960/download?download_frd=1)   // a valid puzzle
  - **Latin_5x5.txt** (https://canvas.wisc.edu/courses/248855/files/20656955/download?download_frd=1)   // a valid puzzle
  - **Latin_4x4_Invalid.txt** (https://canvas.wisc.edu/courses/248855/files/20656953/download?download_frd=1)   // an invalid puzzle - fails both alphabet and rows/columns tests
  - **Latin_4x4_RC_Invalid.txt** (https://canvas.wisc.edu/courses/248855/files/20656954/download?download_frd=1)   // an invalid puzzle - fails the rows/columns test
  - **Latin_5x5_Alphabet_Invalid.txt** (https://canvas.wisc.edu/courses/248855/files/20656956/download?download_frd=1) // an invalid puzzle - fails the alphabet test

Compiling and running

```
>>> gcc -g -o latin_square latin_square_functions.c latin_square_main.c -m32 -Wall
>>> ./latin_square Latin_4x4.txt
```

Strategy

Write your code in small pieces and test each line written by printing out a message. This technique is called scaffolding. Remove the debugging messages after you verify the success of your code.

Start working in main first and transfer working code into the functions when ready.

Hardcode a small puzzle board to become familiar with working with the latin_square data structure. Get it to print with the provided Print_Latin_Square(...) function.

Use array notation with [] to begin, but then after your code is working upgrade to use pointers instead to access the arrays.

Resources

See sections 7.5 and 7.7 of K&R for help working with Files

DynamicMemoryAllocation.pdf **(https://canvas.wisc.edu/courses/248855/files/20477939/download?download_frd=1)**

Chapter 5 of K&R for help with pointers.

Check out the **Matrix Transpose Example** **(https://www.youtube.com/watch?v=4Q6BqcioXEY)** beginning at about 55 minutes.

Turn in

Upload your latin_square_functions.c file to Canvas. Do not upload latin_square_main.c. Double-check to make sure you turned in the correct file.

Style

1. Use meaningful variable names. Either use underscores for multi-word variables or CamelCase. Be consistent.
2. Be consistent with capitalization. For example, capitalize function names, use all caps for structs, enums, and #defines, and use lower case for variables.
3. Indent to match scope. Use four spaces instead of tabs.
4. Organize your code into sections as follows:
    1. #include <>
    2. #include ""
    3. #defines
    4. Data Types (e.g., structures and typedef)
    5. Global variables – (use global variables only when necessary)
    6. Function Prototypes
    7. Code - main() should be either the first or last function
5. Comments (It can be helpful to write the comments first)
    1. Describe what each block of code is trying to do.
    2. Good variable names may make this description easier (or potentially unnecessary).
    3. For functions, describe the purpose of the function and indicate the purpose of input parameters, output parameters, and the return value.
6. Multi-statement lines
    1. Should include statements that logically go together.
    2. For example: if (ptr == NULL) { printf(“error message\n”); exit(1); }