עבודה 2 מבוא למערכות הפעלה

מגישים: חן פרידמן ולנה נסירוב

חן פרידמן: 208009845
לנה נסירוב: 321790891

לפי בקשת המתרגל אייל,
הפרוייקט של ה Hackathon נמצא בקישור הבא:
https://drive.google.com/file/d/1SsstVYJ3ixLeuk-uNW2BUpYMnXKzPc_t/view?usp=sharing

שאלה 1

a. כן, אפשר ליישם counting Semaphore באמצעות Mutex ב-#C. counting Semaphore הוא סנכרון פרימיטיבי השולט בגישה לקבוצה מוגבלת של משאבים. הוא שומר על ספירה המייצגת את מספר המשאבים הזמינים. כאשר thread רוצה לרכוש משאב, הוא בודק את ספירת Semaphor. אם הספירה גדולה מאפס, ה-thread יכול לרכוש את המשאב. אחרת, ה-thread ממתין עד שיתפנה משאב.

הנה יישום של counting Semaphore באמצעות Mutex ב-#C:

```
using System.Threading;

1 reference
class MySemaphore
{
    private Mutex mutex;
    private int count;

    0 references
    public MySemaphore(int starting, int max)
    {
        mutex = new Mutex();
        count = starting;
        MaxCount = max;
    }

    2 references
    public int MaxCount { get; }

    0 references
    public bool WaitOne()
    {
        mutex.WaitOne();

        if (count > 0)
        {
            count--;
            mutex.ReleaseMutex();
            return true;
        }
        else
        {
            mutex.ReleaseMutex();
            return false;
        }
    }

    0 references
    public bool Release(int num = 1)
    {
        mutex.WaitOne();

        if (count + num <= MaxCount)
        {
            count += num;
            mutex.ReleaseMutex();
            return true;
        }
        else
        {
            mutex.ReleaseMutex();
            return false;
        }
    }
}
```

b. כן, אפשר להשתמש בסמפורים כדי להבטיח שההצהרות S2,S1 ו-S3 יבוצעו בסדר הספציפי S1,S2,S3. הנה דרך אחת להשיג זאת באמצעות סמפור:

צור שני סמפורים, "synch1" ו-"synch2", שניהם מאותחלים ל-0.

```
T1:
// Code #1 S1;
signal(synch1);
T2:
wait(synch1);
// Code #2 S2;
signal(synch2);
T3:
wait(synch2);
// Code #3 S3;
```

c. האלגוריתם של דקר והפתרון של פיטרסון הם שניהם אלגוריתמים קלאסיים המשמשים לפתרון בעיית הסעיפים הקריטיים בתכנות במקביל. להלן השוואה בין שני האלגוריתמים, יחד עם היתרונות והחסרונות שלהם:

<u>האלגוריתם של דקר:</u>
האלגוריתם של דקר הוא אחד הפתרונות המוצעים המוקדמים ביותר לבעיית הסעיפים הקריטיים. זוהי דוגמה לאלגוריתם של המתנה פעילה (busy-waiting), כלומר הוא מסתמך על בדיקה מתמשכת של תנאי בלולאה עד לעמידה בתנאי. באלגוריתם של דקר מעורבים שני תהליכים, ולכל תהליך יש דגל משלו כדי לציין את כוונתו להיכנס לקטע הקריטי.

יתרונות האלגוריתם של דקר:
1. פשטות: האלגוריתם של דקר יחסית פשוט להבנה ויישום.
2. הגינות: האלגוריתם מבטיח הגינות במובן זה שאם שני התהליכים ינסו להיכנס לקטע הקריטי בו זמנית, הם בסופו של דבר יתחלפו.

החסרונות של האלגוריתם של דקר:
1. הפרת מניעה הדדית: אם שני התהליכים נמצאים בקטע הקריטי שלהם בו זמנית, מאפיין המניעה ההדדית מופר.
2. המתנה פעילה: האלגוריתם משתמש בהמתנה פעילה, כאשר תהליך בודק ללא הרף מצב בלולאה עד שהוא יכול להיכנס לקטע הקריטי. זה יכול לבזבז מחזורי CPU ובדרך כלל אינו יעיל.

<u>האלגוריתם של פיטרסון:</u>
הפתרון של פיטרסון הוא אלגוריתם ידוע נוסף לפתרון בעיית הסעיפים הקריטיים. זה גם כרוך בשני

תהליכים ומשתמש במשתנים משותפים כדי לשלוט בגישה לקטע הקריטי. הפתרון של פיטרסון משתמש במושג משתני תור כדי לתאם בין התהליכים.

יתרונות הפתרון של פיטרסון:

1. מניעה הדדית: הפתרון של פיטרסון מבטיח שרק תהליך אחד יכול להיכנס לקטע הקריטי בכל פעם, תוך שמירה על מניעה הדדית.
2. התקדמות: האלגוריתם מבטיח התקדמות, כלומר אם תהליך רוצה להיכנס לקטע הקריטי, הוא יקבל בסופו של דבר את ההזדמנות לעשות זאת.

החסרונות של הפתרון של פיטרסון:

1. מוגבל לשני תהליכים: הפתרון של פיטרסון תוכנן במיוחד עבור שני תהליכים. זה לא ניתן להרחבה בקלות ליותר משני תהליכים.
2. המתנה פעילה: בדומה לאלגוריתם של דקר, הפתרון של פיטרסון מסתמך גם על המתנה פעילה, מה שעלול להוביל לחוסר יעילות ובזבוז משאבים.

בסך הכל, גם לאלגוריתם של דקר וגם לפתרון של פיטרסון יש יתרונות וחסרונות. בעוד שהאלגוריתם של דקר פשוט יחסית, הוא סובל מהפרות של מניעה הדדית ומחוסר היעילות של המתנה פעילה. הפתרון של פיטרסון מטפל בבעיות אלו אך מוגבל לשני תהליכים ועדיין משתמש בהמתנה פעילה.

שאלה 3

Which types of lock does your object use?

My SharableSpreadSheet uses mutex and SemaphoreSlim.

How many lock does your program use?

```
//readers
private Mutex[] _rowReadMutexLock;
private Mutex[] _colReadMutexLock;

//writers
private Mutex[] _rowWriteMutexLock;
private Mutex[] _colWriteMutexLock;

//Table lock
private SemaphoreSlim _semaphoreUsers;
private Mutex _mutexLockTable;

//rows and cols exchanging locks
private Mutex _rowExchangingLock;
private Mutex _colExchangingLock;
```

it uses 2*rows+2*cols +2 mutexes and 1 `SemaphoreSlim`.

for each row and column there are two lock- reading and writing.

Add to your report.pdf detailed description of your internal object design.

1. `Mutex[] _rowReadMutexLock`
2 `Mutex[] _colReadMutexLock`

An array in the size of the number of rows/cols. Each cell in the array is a mute. We are using these mutexes when we want protect the counter of the readers in each row/col.
The array that holds the number of readers in each row/coll called:
`int[] _rowReadIntCounter and int[] _colReadIntCounter`.

3. `Mutex[] _rowWriteMutexLock`
4. `Mutex[] _colWriteMutexLock`

A Mutex array in the size of the number of rows/cols. Each cell in the array is a mutex.
We are using these mutexes when we want to lock the write in the row/col.

5. `Mutex _rowExchangingLock`
6. `Mutex _colExchangingLock`

We are using these semaphore when we want to prevent rows/cols changing or exchanging in the spreadsheet.

7. `SemaphoreSlim _semaphoreUsers`
8. `Mutex _mutexLockTable`

SemaphoreSlim initialized to (nUsers, nUsers). We are using the SemaphoreSlim lock the whole table when needed in some functions. The mutex responsible to cause all the threads wait while we create a new SemaphoreSlim. When we need to lock the whole table, we need to acquire all slots for the users in the semaphore and wait for all the threads to finish their work.
when table lock method getting used, the _mutexLockTable is a mutex that gets locked and stops reading and writing methods to access the table.
all the functions try to acquire the mutexLockTable and then release it in order to check the whole table isn't locked.

read only methods:

- getCell
- searchString
- searchInRow
- searchInCol
- searchInRange
- findAll
- search
- searchInRange
- searchInCol
- searchInRow
- getSize

writing methods:

- setCell
- exchangeRows
- exchangeCols

Locking the whole spreadsheet methods:

- addRow
- addCol
- save
- load
- setAll

Diagram:
legend: blue – locked for writing, open to reading.
red – locked for reading and writing
setCell(2,2,newStr):

testCell00	testCell01	testCell02	testCell03	testCell04
testCell10	testCell11	newStr	testCell13	testCell14
testCell20	testCell21	testCell22	testCell23	testCell24
testCell30	testCell31	testCell32	testCell33	testCell34

exchangeRows(1,2):

testCell00	testCell01	testCell02	testCell03	testCell04
testCell20	testCell21	testCell22	testCell23	testCell24
testCell10	testCell11	testCell12	testCell13	testCell14
testCell30	testCell31	testCell32	testCell33	testCell34

exchangeCols(3,4):

testCell00	testCell01	testCell02	testCell04	testCell03
testCell10	testCell11	testCell12	testCell14	testCell13
testCell20	testCell21	testCell22	testCell24	testCell23
testCell30	testCell31	testCell32	testCell34	testCell33

searchInRange(1,2,1,2,testCell11):
searchString(testCell11):
searchInRow(1,testCell11):
searchInCol(2, testCell11):

testCell00	testCell01	testCell02	testCell03	testCell04
testCell10	testCell11	testCell12	testCell13	testCell14
testCell20	testCell21	testCell22	testCell23	testCell24
testCell30	testCell31	testCell32	testCell33	testCell34

setAll(strOld,strNew,true)

testCell00	testCell01	testCell02	testCell03	testCell04
testCell10	testCell11	testCell12	testCell13	testCell14
testCell20	testCell21	testCell22	testCell23	testCell24
testCell30	testCell31	testCell32	testCell33	testCell34

About locks in the different functions:
Synchronization between the functions: in our solution we used the idea of the reader's writes problem from the lecturer:
Only one writer is allowed to write simultaneously in the shared data.
We allow multiple readers to read while there is no writing in the shared data.
The first reader will lock the writing mutex and the last reader will release it.

About the reading only functions:
- **getCell**: in this function multiple readers can read the data of the cell, after both row and column of the specific cell got reading permission.
- **searchString**: in this function we call the searchInRange function in the range of all the spreadsheet.
- **searchinRange**: in this function we use the search method and return the first result. If there is not result, search method will return <-1,-1>
- search: multiple readers can read the data of the cell, after getting reading permission. in this function exchangeRows and exchangeCols is not allowed and prevented by locking the mutexs that responsible on that. No need to get column reading lock, because the row is enough.
- **searchInRow**: in this function multiple readers can read the data of the cell, after getting reading permission. in this function exchangeCols is not allowed and prevented by locking the mutex that responsible on that, exchangingRows is allowed.
- **searchInCol**: the same as searchInRow but the opposite.
- **findAll**: this function uses a helper function that is similar to searchInRange function that continues the search even after finding one element. Also this helper function consider or not consider case sensitivity.
- **getSize**: in this function we are reading the size of tows and cols in the spreadsheet.

About the writing functions:

- **setCell**: in this function we lock the specific row and col and performing write in the cell.
- **exchangeRows**: in this function first we need to lock the option of exchanging cols. Then we will lock the two rows we need to exchange. Performing exchanging and then release all locks.
- **exchangeCols**: same as exchangeRows function.
- **setAll**: this function uses a helper function that is similar to findAll and searchInRange function that continues the search even after finding one element. Also this helper function consider or not consider case sensitivity. Before setting a cell, we will lock the cell.

About the lock all spreadsheet functions:

- In the: addRow, addCol, save and load functions we will lock the whole spreadsheet.

For adding new row we will create new temp spreadsheet and copy the old data in addition to the new row/call that will be empty.
For adding new col we will resize all the rows array.
In the save and load functions we will save in a certain way the content of the spreadsheet and we will load it in the same certain way when needed.

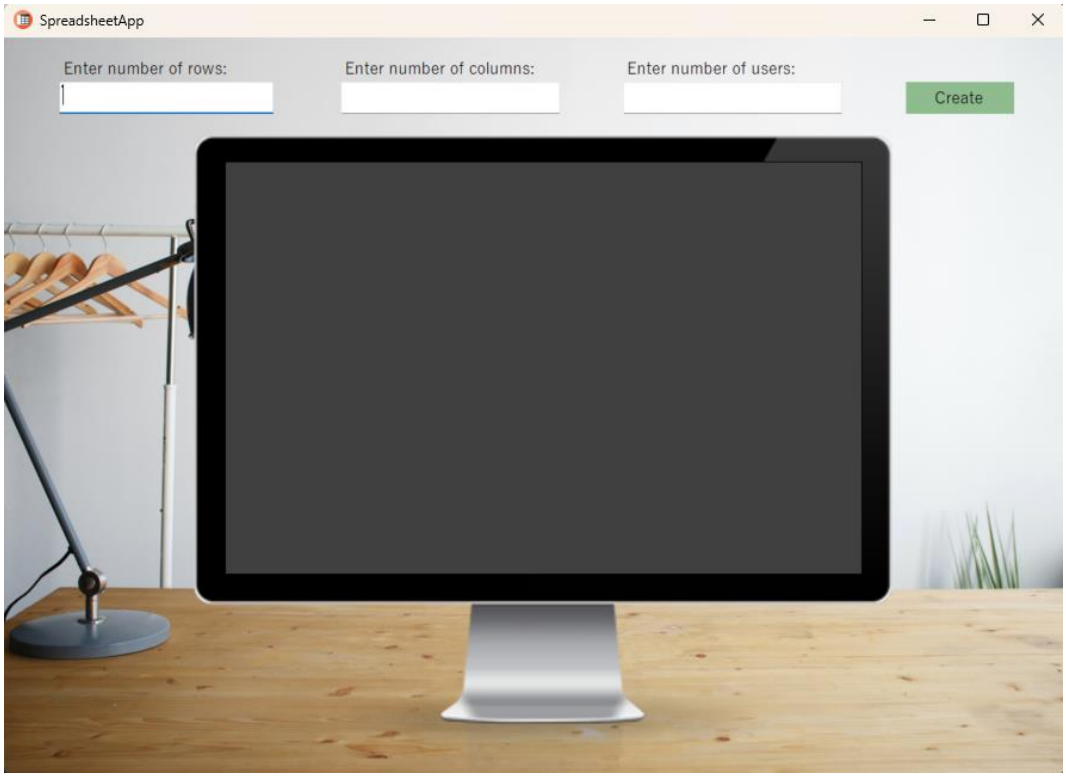
- Each function that doesn't lock the table, will ask permission to enter to table by starting at:

```
this._mutexLockTable.WaitOne();
this._semaphoreUsers.Wait();
this._mutexLockTable.ReleaseMutex();
```

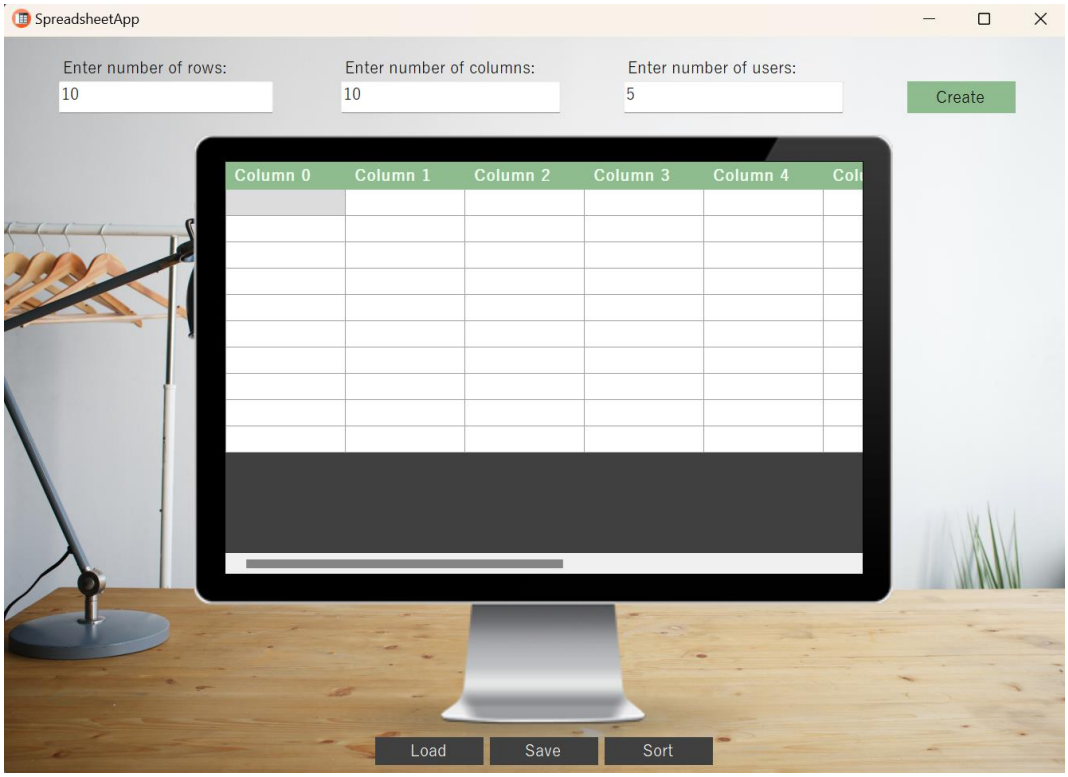
When the function ends we will release the `_semaphoreUsers`.

שאלה 5

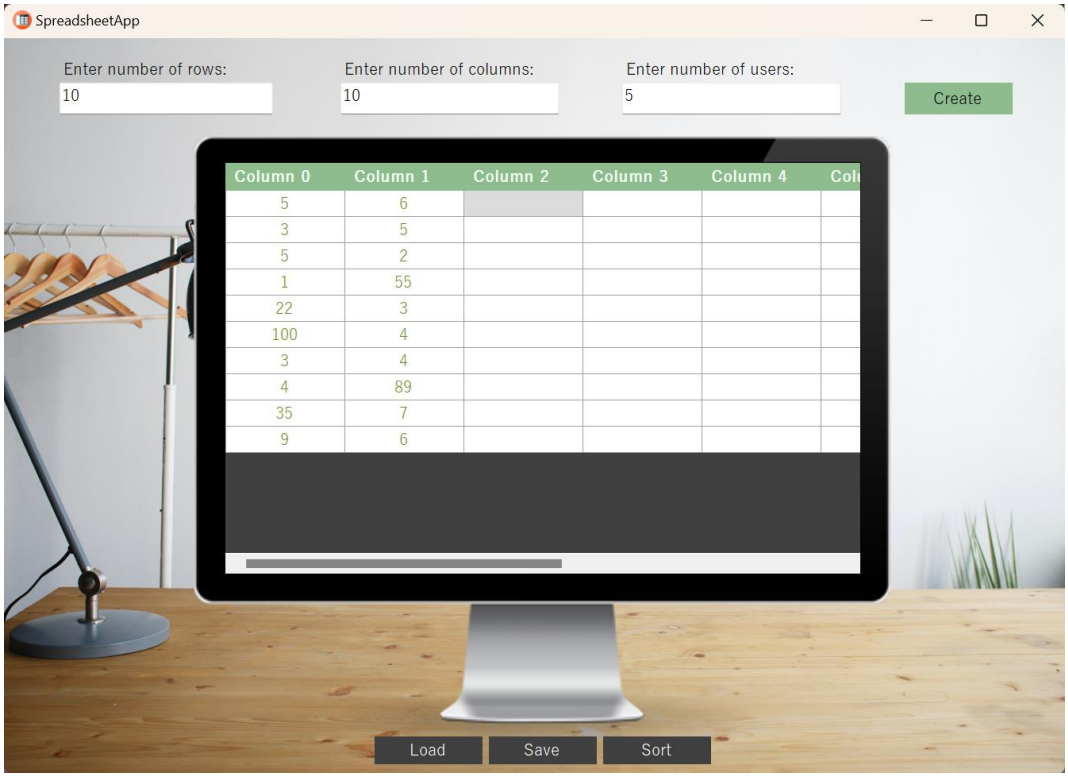
בהרצת התוכנית זה המסך שמופיע:



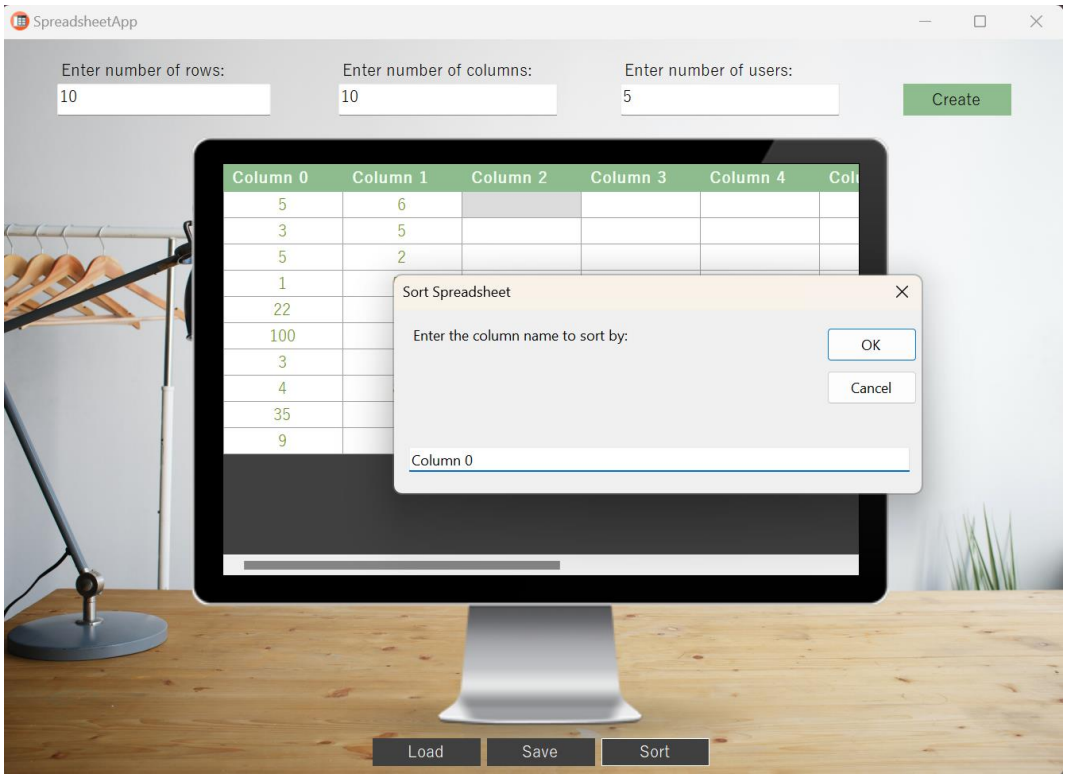
בלחיצה על **Create** נוצר Spreadsheet חדש:



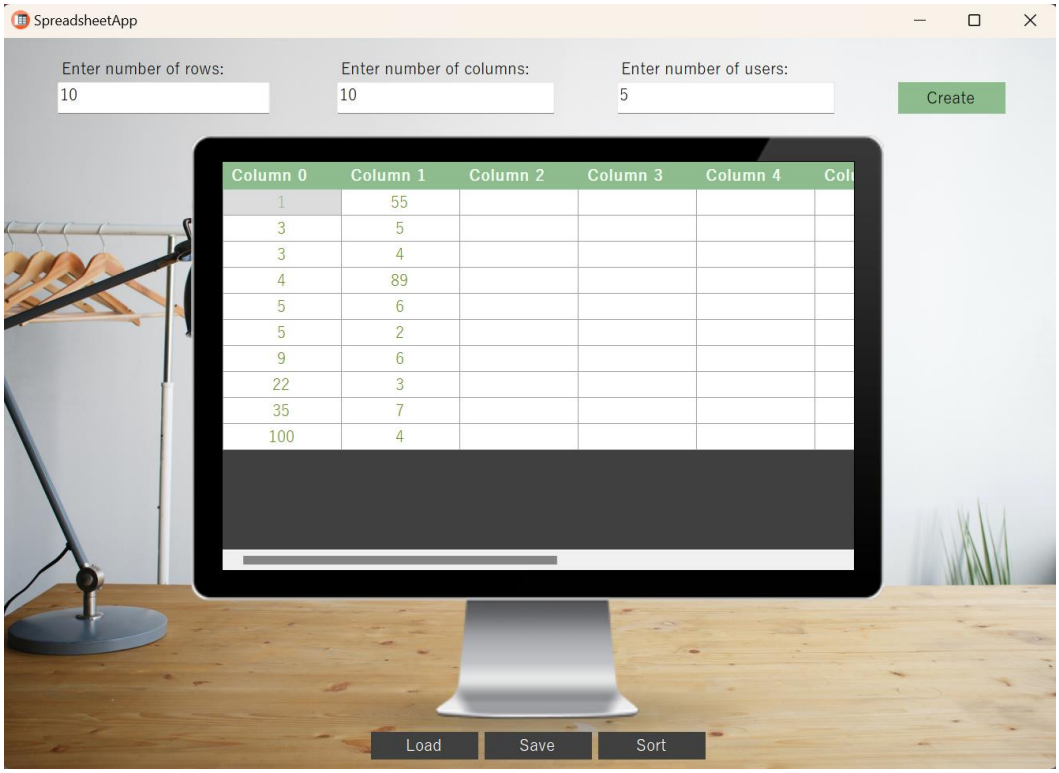
לאחר הכנסת ערכים:



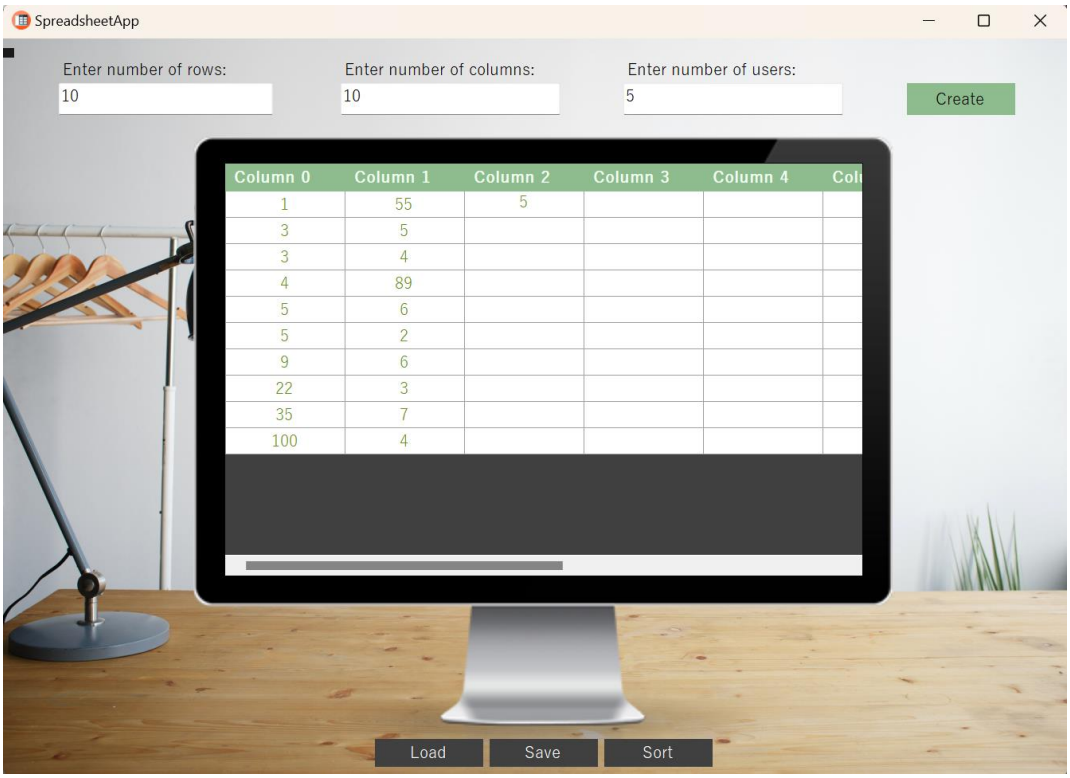
בלחיצה על **Sort** יש להכניס את שם העמודה שנרצה למיין:



בלחיצה על **OK** נקבל את העמודה ממוינת:



בלחיצה על **Save** הSpreadsheet הנוכחי ישמר.
נוסיף ערך נוסף בלי ללחוץ על Save:



בלחיצה על **Load** נקבל את הSpreadsheet השמור האחרון: