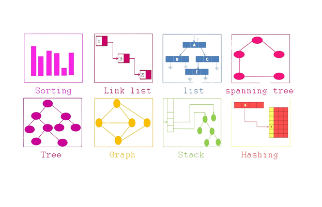
תרגול 1  
מבני נתונים

# רקע

כדי לפתוח את השנה החדשה, הכנו לכם תרגיל מגניב שיתרגל כמה מהדברים שראינו בשיעור.  
תזכורת קצרצרה...  
   
את השיעור התחלנו בלהציג **שפה חדשה** – **C++** וכמה פקודות חדשות ששונות מאיך שהיינו רגילים בשפת C.   
  
אחר כך עברנו על נושא **מבני נתונים**, שם נתנו כמה דוגמאות למבני נתונים ישנים שכבר הכרנו, וגם הצגנו מבני נתונים חדשים – תור ומחסנית. ראינו שאפשר להתייחס למבנה נתונים כאל משהו מופשט, ושלמעשה אפשר לממש את הרעיון שמאחורי כל מבני נתונים בכמה דרכים שונות. בשיעור עצמו ראינו דוגמה קצרה של מחסנית שמוממשת באמצעות מערך.   
  
בסוף השיעור **למדנו על GIT** ושזה אחלה דרך לשמור את הקוד שלנו ולעקוב אחריו... המדריך/ה גם הזכיר שיש **בדיקות (טסטים) אוטומטיות** שרצות באתר GitHub...  
  
בקיצור, יש לנו לא מעט דברים שעליהם נעבור בתרגיל הזה... 😏  
זה עשוי להיראות קצת עמוס, אבל זה רק בגלל שיש הרבה דברים שנלמד פעם ראשונה היום, ואחר כך נעשה אותם בקלות בשאר התרגילים...

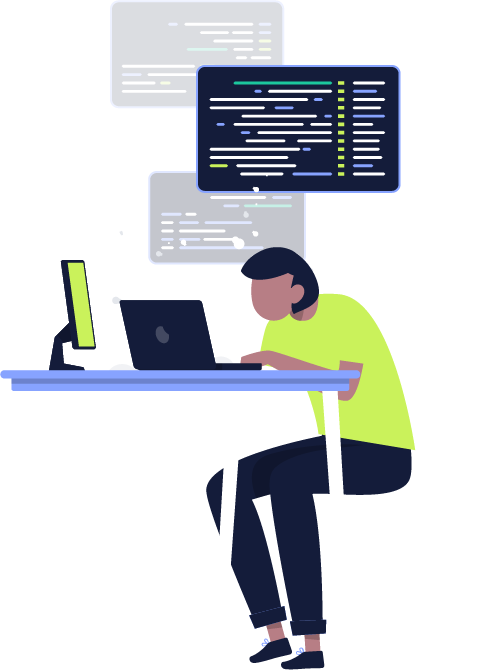


# מטרה

**בתרגיל נבנה כמה מבני נתונים מוכרים, ונשתמש בשפת C++ כדי לעשות זאת.**

אלה השלבים שתעברו:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **GIT (הכנת ה-repo)** | **תור  (Queue)** | **רשימה מקושרת (Linked List)** | **מחסנית (Stack)** | **Utils** | **בדיקות אוטומטיות** |
| נעשה כמה הכנות ב-GIT לפני שנתחיל לעבוד על התרגיל עצמו. | נממש תור באמצעות מערך. | נממש רשימה מקושרת. | נממש מחסנית שמבוססת על רשימה מקושרת. | נהפוך מערך בעזרת המחסנית שכתבנו | נוודא שהבדיקות האוטומטיות עוברות ב-GIT |

  
  
נתרגל מיומנויות חשובות:

* היכרות עם שפת C++ וכמה פקודות חשובות.
* נעלה את הקוד שלנו לאתר GitHub.
* נממש מבני נתונים מוכרים.
* "נחזור לכושר" אחרי שהרבה זמן לא תיכנתנו 😎

את התרגיל צריך להגיש ב-GIT: [לינק להוראות שימוש ב-GIT.](#_הגשה_ב-GIT)

כדאי לקרוא גם [דגשים לתכנות נכון](#_דגשים_לתכנות_נכון).

המסמך אמנם ארוך,   
אבל הרבה ממנו הוא חומר קריאה שנכתב כדי שנוכל   
לתרגל את הדברים **ביחד** – ולא לזרוק אתכם/ן למים לבד.   
אל תיבהלו מהאורך! כתבנו אותו כדי לעזור לכם 🤗

**בהצלחה יא אלופות ואלופים!**

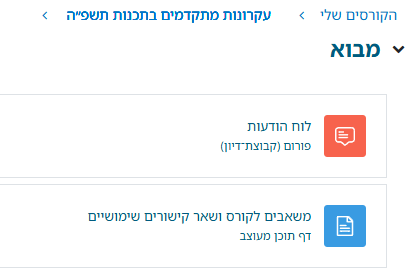
“*Practice Makes Perfect”*

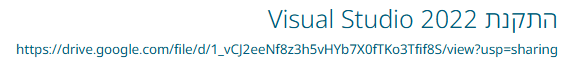
# שלב 0: יוצרים סביבת עבודה

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GIT (הכנת ה-repo)** | **תור  (Queue)** | **מחסנית (Stack)** | **Utils** | **בדיקות אוטומטיות** |

### משימה 1 – התקנת **Visual Studio 2022**

אם לא עשיתם/ן את זה עדיין – היכנסו לכיתת ה-MOODLE עם הקוד שקיבלתם/ן מהמדריך/ה ופעלו ע"פ מדריך ההתקנה של VS2022.   
הוראות התקנה תוכלו למצוא בלשונית ה-משאבים בחלק של "מבוא"





### משימה 2 – סנכרון המחשב עם הפרויקט באתר **GitHub**

כדי להתחיל לעבוד עם GIT כמו שצריך, אנחנו קודם צריכים לדאוג שהמחשב שלנו יהיה מתואם עם אתר GitHub. במילים אחרות נרצה ששינויים שאני עושה במחשב שלי יוכלו להעלות לאתר GitHub, ובשביל זה נעשה כמה צעדים:

1. ניכנס לאתר <Github.com>, ונתחבר עם המשתמש שלנו.
2. ניכנס ל- **Your repositories**ונפתח את התרגיל הראשון.
3. נבחר תיקייה במחשב שלנו שנוח לנו לעבוד ממנה, נפתח מתוכה את ה-**Git Bash** ואז נבצע את הפקודה **git clone**.
4. אל התיקייה של הפרויקט נוסיף קובץ **.gitignore** (שימו לב שהשם של הקובץ מתחיל בנקודה!) מומלץ לקחת את ה-.gitignore הבא, הוא מתאים לכל אורך הקורס.  
   <https://github.com/github/gitignore/blob/main/VisualStudio.gitignore>
5. חשוב לוודא שקובץ ה-**gitignore** שלנו עובד טוב ומתעלם מהקבצים.   
   לצורך כך צרו קובץ בשם test.exe וגם תיקייה בשם Debug בתיקיית הפרויקט שלכם.   
   הריצו git status ותוודאו שהוא נקי! אם הקובץ test.exe מופיע תחת Untracked Files – **סימן שעשיתם משהו לא נכון**. חזרו על ההוראות של יצירת קובץ **".gitignore"** פעם נוספת.  
   אם הכול תקין – **כל הכבוד!** מחקו את הקובץ והתיקייה ותמשיכו הלאה.  
     
   נשמור את השינויים בתוך commit עם ההודעה "Initial Commit".
6. נדחוף את השינויים שעשינו ל-Repo המרוחק.

* ניכנס ל-Repo שלנו באתר, ונוודא שהקבצים הועלו אליו בהצלחה.  
  **מסתבכים?** כדאי לצפות [בסרטון הבא.](https://drive.google.com/drive/folders/1uL39MYoxh_x7Fzezp5eEx_a_T8fb2h59?usp=drive_link), או לעיין [בתרגולים הבסיסיים על גיט וגיטהב](https://drive.google.com/drive/u/1/folders/1MF_6vILTclRA_t5f5MsbR-R4NU-Fkrxp)

⚠חשוב! ⚠

לפני שממשיכים לשלב הבא, עברו על המסמך הבא שמסביר על מערכת הבדיקות האוטומטיות:

✳ [קישור להסבר על GitHub ומערכת הבדיקות](https://drive.google.com/file/d/1xt3NLDVFUxm-F0eaSdZPzq_mNLnfBhY8/view?usp=sharing)✳

### משימה 3 – סנכרון המחשב עם הפרויקט באתר **GitHub**

פתחו פרויקט חדש ב-Visual Studio בתוך התיקייה של ה-repo. שם הפרויקט יהיה "**DataStructures**" או שם הגיוני אחר.

וודאו שסימנתם/ן את ה-checkbox שנמצא במסך יצירת הפרויקט:

A screenshot of a computer program

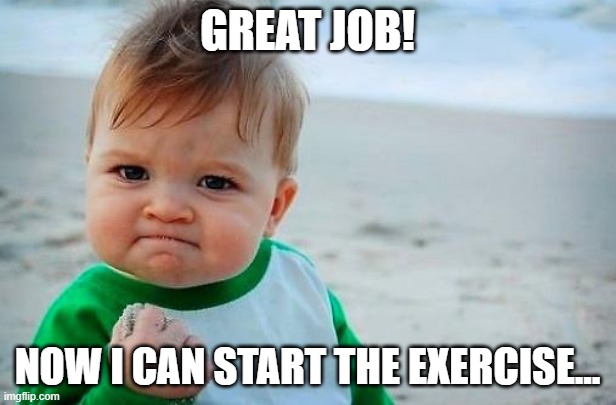
Description automatically generated☑ “Place solution and project in the same directoy” ☑

1. נעשה add לכל הקבצים החדשים שנוצרו.
2. נעשה commit לשינויים עם הודעה אינדיקטיבית.  
   לדוגמה: "Added Visual Studio files".
3. נדחוף את השינויים שעשינו ל-Repo המרוחק.

### בסיום שלב 0 אמור להיות לנו:

* Repo שאליו אנחנו מחוברים באתר GitHub (שהמדריך/ה פתח/ה לנו).
* תיקייה על המחשב שאליה עשינו clone מהשרת.
* בתוך התיקייה – יש לנו קובץ בשם .gitignore , ותיקייה של פרויקט Visual Studio.
* כל השינויים שעשינו בתיקייה – נמצאים גם בתוך ה-Repo על השרת (וודאו שהקבצים אכן נמצאים שם!)

**הכול פיקס? מעולה! אפשר להתחיל ☺**



ואם הכול לא פיקס? לא נורא...  
עברו שוב על השלבים במסמך, ועל מסמך לימוד העצמי.  
עברו על התרגיל המקדים שנשלח לפני תחילת שנה, זה מאוד יעזור.

מעבר לזה תמיד אפשר לפנות למדריך/ה שלנו, אלו דברים מאוד חשובים בגלל שכל תרגיל בהמשך השנה ייפתח בדיוק באותה צורה, אם נבין את זה טוב זה יקל עלינו המון בהמשך... 💪

# שלב 1: מימוש תור באמצעות מערך

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GIT (הכנת ה-repo)** | **תור  (Queue)** | **מחסנית (Stack)** | **Utils** | **בדיקות אוטומטיות** |

בחלק זה נממש ADT (Abstract Data Type) מסוג תור המכיל מספרים שלמים וחיוביים.   
הסבר על הדרישות מה-ADT הזה מופיעות במצגת של השיעור (שאליה אפשר להיכנס מה-MOODLE).

### משימה 1 – מימוש תור באמצעות מערך

יש לממש את הפונקציות המופיעות בקובץ ה-header שקיבלתם - Queue.h

יש לממש את התור **בעזרת מערך** **המוקצה דינמית** בעת האתחול.

הפעולה enqueue - היא הפעולה המכניסה איברים לתור.

הפעולה dequeue -היא הפעולה המוציאה איברים מהתור.

initQueue & cleanQueue - פונקציות האתחול והניקוי.

הפעולות isEmpty ו-isFull - מחזירות משתנה bool שאומר אם התור ריק/מלא.

ניתן להוסיף פונקציות עזר נוספות כרצונכם/ן.

**שימו** ❤מותר (וצריך) להוסיף הגדרות לקובץ ה-Header שניתן ע"י מגשימים (לדוגמה אנחנו צריכים להוסיף משתנים לתוך ה-struct שמייצג את ה-Queue).   
אבל, אסור לשנות את החתימות של הפונקציות הנתונות בשום אופן ולא את שם המבנה!

**תזכורת:** יש לממש את הפונקציות בקובץ נפרד מקובץ ההצהרות.

בדקו את המימוש שלכם ע"י כתיבת main1.cpp . דאגו להעלות את קובץ ה-main ל-repo.

### משימה 2 – העלאת הקוד ל-GIT

**הוסיפו** את הקבצים הבאים (באמצעות הפקודה **git add**):

1. Queue.h
2. Queue.cpp

עשו להם **commit** עם תיאור ברור, ודחפו אותם ל-Repo המרוחק בעזרת **push**.  
בנוסף, דאגו להעלות את ה-main1.cpp שבו השתמשתם/ן כדי לבדוק את תקינות התור.

# שלב 2: מימוש מחסנית באמצעות רשימה מקושרת

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GIT (הכנת ה-repo)** | **תור  (Queue)** | **מחסנית (Stack)** | **Utils** | **בדיקות אוטומטיות** |

בשיעור ראינו דרך אחת אפשרית לממש מחסנית – באמצעות מערך בעל גודל קבוע (שנקבע באופן דינמי בעת יצירת המחסנית).   
כאמור, מחסנית היא ADT (טיפוס נתונים אבסטרקטי), כלומר זו הגדרה כללית של ממשק ולא של דרך מימוש ספציפית. יש יותר מדרך אחת לממש מחסנית. בחלק הזה אנחנו נממש מחסנית **באמצעות רשימה מקושרת**.

### משימה 1 – יצירת רשימה מקושרת

ממשו מבנה שמייצג רשימה מקושרת (של מספרים שלמים וחיוביים) ופונקציות הוספה והסרה מהרשימה.

פונקציית ההוספה תמיד תוסיף לראש הרשימה.

פונקציית ההסרה תמיד תסיר מראש הרשימה.

**הערה:**  חשבו האם יש להעביר מצביע לפונקציות הללו? האם יש להעביר מצביע למצביע?

ניתן להוסיף פונקציות נוספות בהתאם לשיקול דעתכם

יש להכין שני קבצים נפרדים:

* LinkedList.h - קובץ header שיכיל את הגדרת המבנה ואת ההצהרות על הפונקציות של הרשימה המקושרת.
* LinkedList.cpp - קובץ שיכיל את מימוש הפונקציות.

ניתן להשתמש בקוד של רשימה מקושרת שכתבתם בקורס מבוא לתכנות בשנה שעברה.

**שימו לב!** יש לעשות התאמות בין C ל-C++. למשל הקצאת זיכרון על ידי new ולא על ידי malloc.

צרו פונקציה ראשית ובדקו שהרשימה המקושרת שמימשתם עובדת כראוי, העלו את הקוד ל-GIT לפני שממשיכים לסעיף הבא...



### משימה 2 – מימוש מחסנית

כעת יש לממש מחסנית אשר מכילה מספרים שלמים וחיוביים. למחסנית חייבות להיות הפונקציות הבאות:

* pop
* push
* initStack
* cleanStack

שנתונות לכם בקובץ Stack.h - ואין לשנות את חתימתן.

עליכם גם להגדיר אילו שדות המבנה של המחסנית מכיל.

**שימו לב!** אין לשנות את שם המבנה (אבל צריך להוסיף משתנים ל-struct).

ניתן להוסיף פונקציות עזר כרצונכם.

המחסנית שתממשו תתבסס על הרשימה המקושרת של הסעיף הקודם (ולא על מערך כפי שהוצג בשיעור). זאת הסיבה שלמחסנית אין גודל מקסימלי.

צרו קובץ main2.cpp ובדקו שהמחסנית שמימשתם עובדת כראוי.

### משימה 3 – העלאת הקוד ל-GIT

* **הוסיפו** את השינויים שעשיתם/ן
* עשו להם **commit** עם תיאור ברור שכל מי שקורא אותו יבין מה עשיתם/ן בגרסת הקוד שעלתה.
* דחפו אותם ל-Repo המרוחק בעזרת **push**.



# שלב 3: היפוך מערך באמצעות מחסנית

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GIT (הכנת ה-repo)** | **תור  (Queue)** | **מחסנית (Stack)** | **Utils** | **בדיקות אוטומטיות** |

בחלק הזה נעשה שימוש בקוד שכתבנו – נהפוך מערך מספרים באמצעות המחסנית שבנינו.

שימו 💜

עבור סעיפים **ג** + **ד** נתון לכם קובץ Utils.h ובו חתימות של שתי פונקציות. אין לשנות את החתימה.

### משימה 1 – מימוש פונקציית reverse

ממשו בקובץ Utils.cpp את הפונקציה:

void **reverse** (int\* nums, int size)

המקבלת מערך של מספרים (כאשר nums הוא המערך, ו-size הוא מספר האיברים במערך) והופכת את הסדר שלהם במערך.

לדוגמא: לאחר הפעלת הפונקציה על מערך הנראה כך: 1,2,3,4 - המערך יראה כך : 4,3,2,1 .

עליכם להיעזר במחסנית שכתבתם בסעיף הקודם לשם כך.

כתבו תכנית ראשית אשר בודקת את הפונקציה הנ"ל.

### משימה 2 – מימוש פונקציית 10reverse

קראו באינטרנט על האובייקט **cin** שנמצא ב-namespace הסטנדרטי – std. אובייקט זה משמש לפעולות קלט (בדומה לאובייקט cout המשמש לפעולות פלט, עליו למדנו בשיעור).

יש להשתמש בו במימוש הפונקציה הבאה:

int\* reverse10 ()

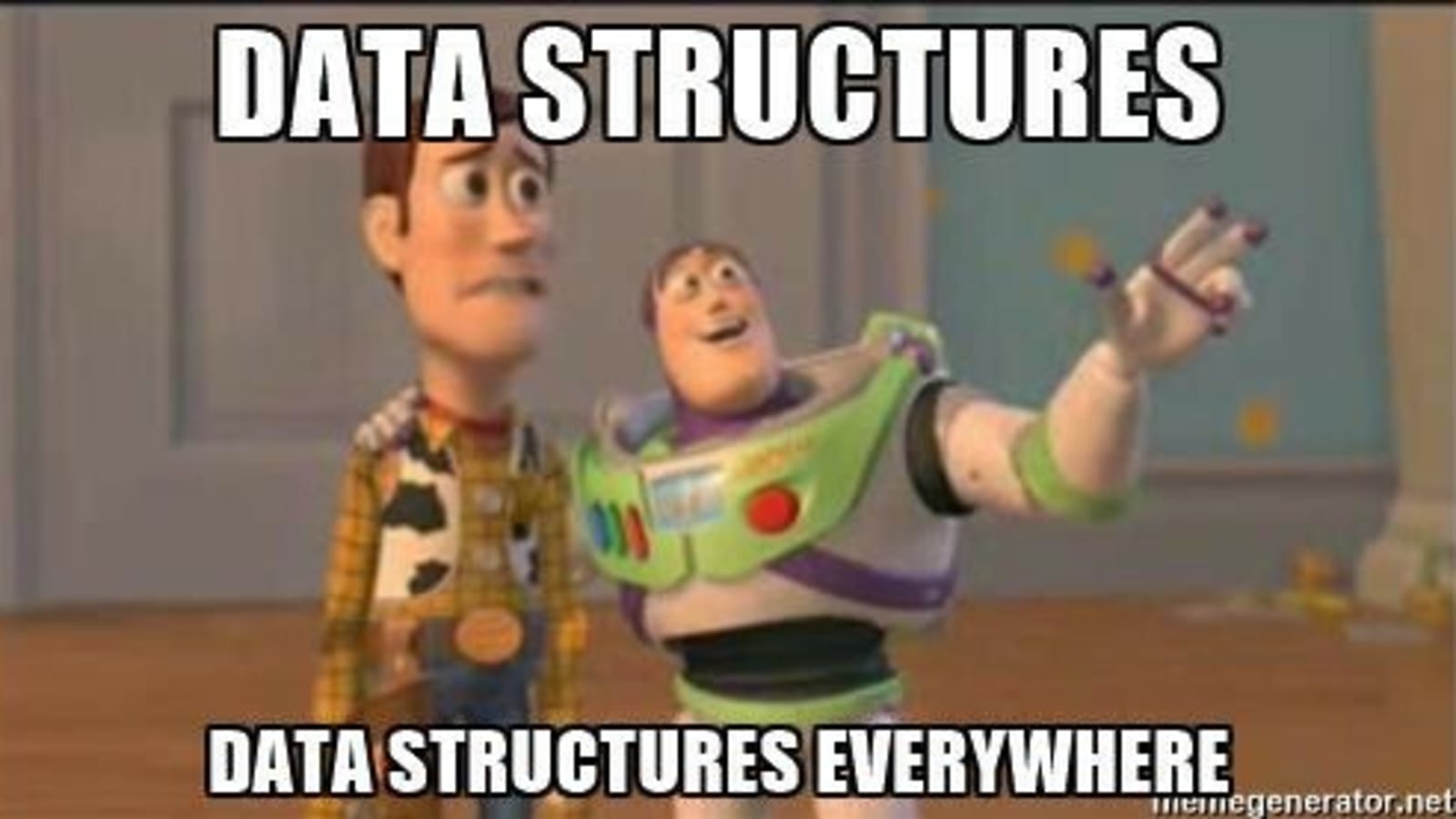
שמוגדרת בקובץ Utils.h. הפונקציה תבקש מהמשתמש להזין עשרה מספרים שלמים (יש לקלוט אותם באמצעות std::cin בלבד). את המספרים יש לשמור במערך. הפונקציה תחזיר מערך שמכיל את המספרים בסדר הפוך ממה שנקלט.

**שימו לב:** המערך צריך להיות מוקצה דינאמית בתוך הפונקציה.

### סיכום חלקים 2 ו-3

וודאו ששמרתם את הקבצים הבאים בתוך ה-Repo שלכם/ן,  
ועשיתם/ן להם **push** ל-Repo המרוחק:

1. LinkedList.h
2. LinkedList.cpp
3. Stack.h
4. Stack.cpp
5. Utils.cpp
6. main2.cpp



# שלב 4: בדיקות אוטומטיות

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GIT (הכנת ה-repo)** | **תור  (Queue)** | **מחסנית (Stack)** | **Utils** | **בדיקות אוטומטיות** |

אחד הדברים שחשוב לוודא אחרי שמעלים קוד ל-GIT, זה שהבדיקות שרצו על הקוד עברו בהצלחה.

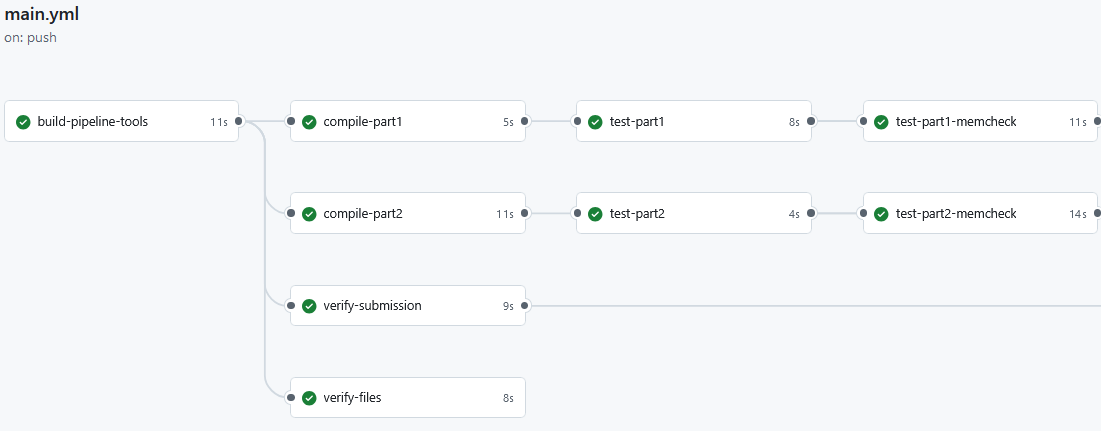
זכרו לעבור על המסמך הבא שמסביר על מערכת הבדיקות האוטומטיות:

✳ [קישור להסבר על GitHub ומערכת הבדיקות](https://drive.google.com/file/d/1xt3NLDVFUxm-F0eaSdZPzq_mNLnfBhY8/view?usp=sharing)✳

ישנם כמה טסטים שצריך לעבור בתרגיל מספר 1:

* **Verify Submission**  
  בודק שהגשנו את התרגיל בצורה נכונה:
  + יש קובץ .gitignore ב-repo
  + שבתוך ה-repo יש פרויקט VS
  + שאין קבצים מיותרים שמונעים פתיחה נוחה של הפרויקט (לדוגמה תיקיית x64 או תיקיית .vs או קובץ vcxproj.users שנוצרים ע"י VS).  
    חשוב לזכור שאם הוספנו .gitignore אז הקבצים האלו לא יעלו ל-repo ואנחנו לא צריכים לעשות כלום.
* **בדיקות קומפילציה**  
  בודקות שהחלקים השונים של התרגיל התקמפלו בהצלחה  
  כדי לוודא שהבדיקות יעברו, אפשר להוריד את הטסטרים מתיקיית ה-Google Drive של התרגיל (שאליה אפשר לגשת דרך ה-MOODLE).
  + **Compile Part 1**מקמפל את הקבצים Queue.cpp, Queue.h והטסטר שיש ב-drive (נקרא Part1Tester.cpp)
  + **Compile Part 2**מקמפל את הקבצים של חלקים 2 ו-3 עם הטסטר שיש ב-drive (נקרא Part2Tester.cpp)
* **טסטים לתרגיל**על ה-GIT ירוצו הטסטים שיש ב-Google Drive, הם יבדקו חלקים שונים בתרגיל ושעמדנו במקרי קצה. **את הטסטים האלה אפשר וכדאי להריץ מקומית על המחשב ולא רק להסתמך על בדיקות המערכת.**
  + **Part 1 – test**בודק את התור שכתבנו
  + **Part 2 – test**בודק את המחסנית שמימשנו, ואת הפונקציה reverse.
* **בדיקות זיכרון**הטסט בודק ששיחררנו את כל הזיכרון שהקצינו דינאמית.  
  כלומר אם עשינו new אז הטסט מוודא שגם עשינו delete מתאים.
* **בדיקות כלליות**יהיו גם בדיקות כלליות שירוצו:
  + **Verify Necessary Exercise Files**בודק שהגשנו את כל הקבצים הנחוצים (כולל קבצי main שבהם השתמשנו לבדוק את הקוד של עצמנו).
  + **VerifySubmissionDate**בודק שהגשנו את התרגיל בזמן.  
    חשוב לזכור שגם אם הגשנו את התרגיל אחרי הזמן והטסט נכשל, זה לא משפיע עלינו, וזה לא ימנע ממנו להעלות משהו נוסף.  
    הבדיקה בסה"כ נועדה ליידע את המדריך והרכזת על הזמן שבו הגשנו את הקוד.

בסיום העבודה, ה-action צריך להיראות ככה:



# שלב בונוס: סיבוכיות זמן ריצה



עברו על המצגת – **"סיבוכיות זמן ריצה"** ובצעו את המשימה הבאה:

בשאלה הראשונה נדרש לממש תור באמצעות מערך. כל מימוש הגיוני יתקבל, והמימוש המצופה הנפוץ ביותר עובד במקרה הכי גרוע בסיבוכיות לינארית - כלומר **(O(N** - כי נדרש מעבר על כל האיברים בתור.

על התרגיל יינתן בונוס אם ההכנסה וההוצאה של איברים תתבצע במספר פעולות קבוע - **(1)O**

הכנסת איברים בכל מקרה צריכה להיעשות בזמן קבוע **(1)O**, אבל הוצאה יכולה להיעשות (ללא בונוס) גם בסיבוכיות לינארית (O(N. הבונוס מתייחס למימוש שמבצע הוצאת איבר ב-**O(1)**.

**בהצלחה!**

# נספחים

## הגשה ב-GIT

* את הפרויקט יש לנהל ב-Git, הפרויקט ייפתח לנו בסוף השיעור ע"י המדריך, אבל חובה להגיש לינק לפרויקט ב-MOODLE (אפשר לעשות comment עם הלינק או להגיש מסמך txt עם הלינק בפנים).
* יש להעלות ל- repositoryאת כל הקבצים הרלבנטיים לתרגיל (קבצי txt, מסמכים, ומשאבים אחרים שבהם השתמשנו).  
  חשוב להעלות את פרויקט ה-Visual Studio השלם ולהתעלם מקבצים לא נחוצים ([הנחיות במסמך הבא](https://drive.google.com/open?id=1Im_zmWD8TZ8kTB5ynGj6_vhMtWM9MY5U)), במידה ולא הועלה הפרויקט השלם, אין להעלות את שאר הקבצים שיוצר Visual Studio – הם רבים מאוד, הם לא מכילים מידע נחוץ להרצת הפרויקט אצל המדריך, ורק יוצרים בלגן.
* הבחירה אילו קבצים להעלות ל-repository נעשית באמצעות הפקודות add ו-rm. אופציה נוספת (מומלצת) היא להוסיף קובץ .gitignore אשר יתעלם מהקבצים הלא נחוצים. במידה ותרצו תוכלו להיעזר ב[סרטון עזר בנושא GIT](https://drive.google.com/drive/folders/1uL39MYoxh_x7Fzezp5eEx_a_T8fb2h59?usp=drive_link).
* כסיימתם/ן, בדקו שניתן להריצ את הפרויקט בקלות – בצעו Clone אל תיקייה במחשב אשר שונה מזו שעבדתם/ן, ותראו שהפרויקט נפתח ע"י לחיצה על קובץ ה-sln ויכול לרוץ בלי בעיה



### דגשים:

* יש לוודא שכל הקבצים הרלבנטיים נוספו ל-repository (באמצעות הפקודה add), במידת הצורך ניתן להוריד קבצים מיותרים (באמצעות הפקודה rm)
* יש לבצע commit עבור כל סעיף, ובנקודות שבהן הוספנו שינויים חשובים (לפי הדגשים שהועברו בכיתה).
* עבור כל commit, זכרו לכתוב הודעה קצרה ואינפורמטיבית, שאפשר יהיה להבין מה היה השינוי בקוד.
* יש לדחוף את הקוד (באמצעות הפקודה push) ל-repository בסיום העבודה שלנו, חשוב שבסיום העבודה שלנו, ובמידה ונפנה למדריך/ה, ב-repository יהיה הקוד המעודכן ביותר.
* במידה ושכחנו או שאנחנו לא בטוחים איך מעלים קובץ, או מתעלמים מקבצים, כדאי לצפות [בסרטון הבא.](https://drive.google.com/drive/folders/1uL39MYoxh_x7Fzezp5eEx_a_T8fb2h59?usp=drive_link), או לעיין [בתרגולים הבסיסיים על גיט וגיטהב](https://drive.google.com/drive/u/1/folders/1MF_6vILTclRA_t5f5MsbR-R4NU-Fkrxp)
* בסיום העבודה יש להגיש לכיתת ה-Moodle קישור ל-repository.  
  **נקודה חשובה מאוד, לא ליצור תיקיות בשם ex1, ex2 ... ex13.** הסיבה לכך היא ש-שרת הבדיקות מייצר תיקיות כאלה כאלה לכל תרגיל בהתאמה ואם הן כבר יהיו קיימות בפרוייקט**, הבדיקות לא ירוצו כלל.**
* **שימו למבנה התיקיות הראשוני של התרגיל, אם יש תתי-תיקיות לסעיפים שונים, תעבדו על אותם תתי-סעיפים בתיקיות האלו. – הבדיקות האוטומטיות מסתמכות על זה.**

### כללי

1. יש לבדוק שכל המטלות מתקמפלות ורצות ב-VS2022. מטלה שלא תעבור קומפילציה אצל הבודק לא תיבדק והניקוד שלה יהיה 0 ☹
2. יש לבדוק שהקוד שכתבתם עובד. יש להריץ בדיקות שלכם ולוודא שהקוד ברמה טובה.
3. כאשר אתם מתבקשים לממש פונקציה, ממשו בדיוק את הנדרש. **אין להוסיף הדפסות** וכדו'. אם הוספתם תוך כדי הבדיקות שלכם הדפסות, אנא דאגו להוריד אותם לפני ההגשה.
4. **להזכירכם!** העבודה היא **עצמית,** ואין לעשות אותה ביחד.
5. על כל שאלה או בעיה יש לפנות למדריך, לפחות **36 שעות** לפני מועד ההגשה.

## דגשים לתכנות נכון

* כדאי לקמפל כל מספר שורות קוד ולא לחכות לסוף! הרבה יותר קל לתקן כאשר אין הרבה שגיאות קומפילציה. בנוסף קל יותר להבין מאיפה השגיאות נובעות.
* כדאי לכתוב פונקציה ולבדוק אותה לפני שאתם ממשיכים לפונקציה הבאה. כלומר, כתבו תכנית ראשית שמשתמשת בפונקציה ובודקת האם היא עובדת כראוי. חישבו על מקרי קצה ונסו לראות מה קורה.
* בכל פעם שאתם מתקנים משהו, זכרו שיכול להיות שפגעתם במשהו אחר. לכן עליכם לבדוק שוב מהתחלה.
* חשפו החוצה רק את הממשק המינימלי הדרוש (minimal API), הגדירו את שדות המחלקה כפרטיים, וכמה שפחות מתודות כציבוריות.