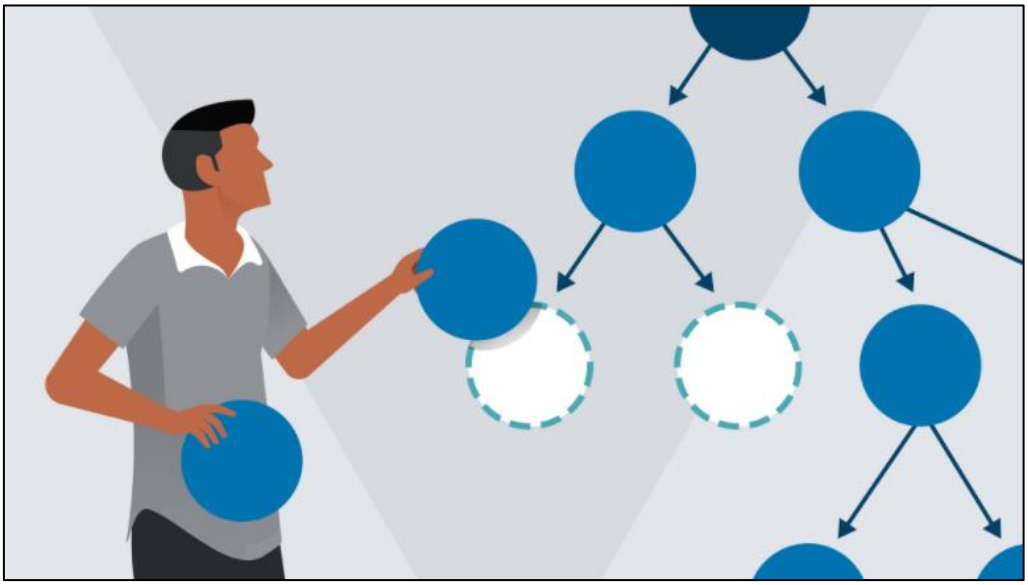
תרגול בית 1 – מבני נתונים



מה נעשה בתרגיל?

כדי לפתוח את השנה החדשה – הכנו לכם תרגיל מגניב, שבו נתרגל ביחד את הנושאים הבאים:

- נתרגל מימוש של מבני הנתונים **תור (Queue)** ו-**מחסנית (Stack)**
- נתרגל את תהליך העבודה עם גיט – פתיחת פרויקט, העלאת קבצים לשרת GitLab, יצירת Commit-ים ועוד.

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

הוראות הגשה

עבודה עם Git

- את הפרויקט ננהל בעזרת גיט. במהלך התרגיל הוספנו הנחיות מדויקות בכל שלב.
- ראשית ניצור קבוצה (group) באתר גיטל ונוסיף אליה את המדריך/ה. יש להיעזר במדריך ליצירת קבוצה בגיטלב כדי ליצור קבוצה בצורה נכונה.
אנא וודאו שהמדריך/ה נוסף/ה לפרויקט כ-Maintainer, בסיום העבודה יש להגיש לכיתת ה-NEO קישור ל-Repository.
- נעלה ל-Repo המרוחק (באתר GitLab) את פרויקט ה-Visual Studio השלם, אבל בלי קבצים מיותרים. נשתמש לשם כך בקובץ gitignore, כפי שלמדנו במסמך הלימוד העצמי.
- במהלך העבודה על התרגיל תוכלו להיעזר ב-סרטוני עזר בנושא GIT שהכנו בשבילכם, וכמובן במסמך הלימוד העצמי שכבר קראתם. ניתן לגשת לסרטונים בלשונית ה-Resources שבכיתת ה-NEO.
- בסיום העבודה נדחוף את הקוד שלנו באמצעות הפקודה push ל-Repo המרוחק, כדי שבאתר GitLab תופיע הגרסה העדכנית ביותר של ההגשה.

הוראות הגשה כלליות

- חשוב לוודא **שכל המטלות מתקמפלות ורצות ב-Visual Studio 2022** – מטלה שלא תעבור קומפילציה אצל הבודק – לא תיבדק ו**הניקוד שלה יהיה 0** ☹
- תבדקו שהקוד שכתבתם עובד. הריצו בדיקות ותוודאו שהקוד שלכם ברמה טובה.
- כאשר אתם מתבקשים לממש פונקציה, ממשו בדיוק את הנדרש. אין להוסיף הדפסות וכדומה. אם הוספתם תוך כדי הבדיקות שלכם הדפסות, אנא דאגו להוריד אותם לפני ההגשה.
- כמו שהיה בשנה הקודמת - העבודה על התרגיל היא כמובן עצמית, ולא ביחד.
- על כל שאלה או בעיה יש לפנות למדריך, לפחות 36 שעות לפני מועד ההגשה - כדי שהוא יספיק לעזור לכם בלי לחץ...

דגשים לתכנות נכון

1. 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.
2. כדאי לכתוב פונקציה ולבדוק אותה לפני שאתם ממשיכים לפונקציה הבאה. כלומר, נכתוב תכנית ראשית שמשתמשת בפונקציה ובודקת האם היא עובדת כראוי. חשבו על מקרי קצה, ונסו לראות מה קורה.

3. בכל פעם שאתם מתקנים משהו, זכרו שיכול להיות שפגעתם במשהו אחר. לכן עליכם לבדוק שוב מהתחלה.

חלק 0 – יוצרים סביבת עבודה

בחלק הזה ניצור את סביבת העבודה שאיתה נעבוד בתרגיל.

שלב 0 – התקנת Visual Studio 2022

אם לא עשיתם זאת עדיין – היכנסו לכיתת ה- NEO ופעלו ע"פ מדריך ההתקנה של VS2019. את המדריך תוכלו למצוא בלשונית ה-Resources, תחת קטגוריית "קישורים שימושיים":

→ התקנת Visual Studio 2022

שלב 1 - פתיחת Repo על השרת

1. ניכנס לאתר Gitlab.com, ונתחבר עם המשתמש שלנו.

2. ניכנס לקבוצה שיצרנו ונפתח Repo חדש בשביל הגשת התרגיל.
אנא בחרו שם הגיוני לתרגיל (לדוגמא Ex1) כדי שלמדריך/ה יהיה קל להבין שמדובר בתרגיל הנוכחי

3. נעשה ל-Repo שיצרנו Clone למחשב שלנו.

4. נוסיף קובץ **.gitignore** (שימו לב שהשם של הקובץ מתחיל בנקודה!) לתיקייה של ה-Repo. קחו את התוכן של הקובץ מהאתר gitignore.io או מכאן

ווידוא תקינות של ה-gitignore

חשוב לוודא שקובץ ה-**gitignore** שלנו עובד טוב ומתעלם מהקבצים. 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.

5. נעשה add לקובץ gitignore שיצרנו.

6. נשמור את השינויים בתוך commit עם ההודעה "Initial Commit".

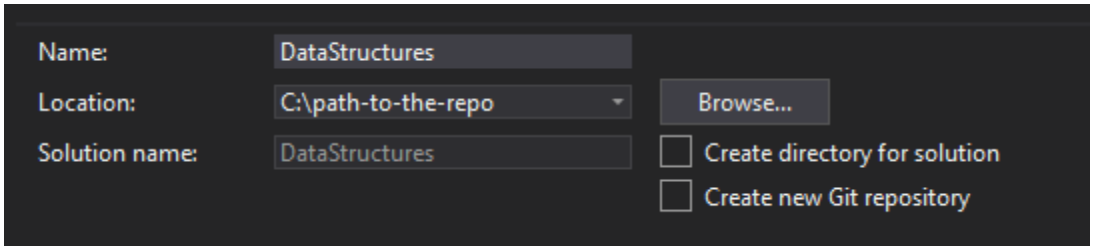
7. נדחוף את השינויים שעשינו ל-Repo המרוחק.

8. ניכנס ל-Repo שלנו באתר, ונוודא שהקבצים הועלו אליו בהצלחה.
מסתבכים? היעזרו במסמך הלימוד העצמי, ובסרטוני ההדרכה.

שלב 2 – פתיחת פרויקט ב-Visual Studio

1. פתחו פרויקט חדש ב-Visual Studio בתוך התיקייה של ה-repo. שם הפרויקט יהיה "**DataStructures**" או שם הגיוני אחר (בבקשה לא לקרוא לפרויקט SUS 🤦).

וודאו שה-checkbox שנמצא במסך יצירת הפרויקט "Create new Git repository" **לא מסומן** (כי כבר יש לנו repo פתוח). הורידו גם את הסימון בתיבה "Create directory for solution, כי כבר יש לנו תיקייה לפרויקט, אין צורך בעוד אחת.



2. נעשה add לכל הקבצים החדשים שנוצרו.

3. נעשה commit לשינויים עם הודעה אינדיקטיבית.
לדוגמה: "Added Visual Studio files".

4. נדחוף את השינויים שעשינו ל-Repo המרוחק.

בסיום חלק 0 אמור להיות לנו:

- Repo שפתחנו על שרת Gitlab, ואליו הוספנו את המדריך שלנו כ-Member (אם פתחנו את הפרויקט בקבוצה המשותפת אז המדריך יתווסף אוטומטית).
- 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 ☺

הכול לא פיקס? לא נורא...
עברו שוב על השלבים במסמך, ועל מסמך לימוד העצמי,
ובדקו מה לא עשיתם כמו שמוסבר במסמך.

חלק 1 – מימוש תור

בחלק זה נממש ADT מסוג תור המכיל מספרים שלמים וחיוביים.
הסבר על הדרישות מה-ADT הזה, מופיעות במצגת.

עליכם 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.

`cleanQueue` & `initQueue` - פונקציות האתחול והניקוי.

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

ניתן להוסיף פונקציות עזר נוספות כרצונכם.

שימו לב! אין 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 main.cpp . דאגו להעלות את קובץ ה-main ל-repo.

משימת גיט

הוסיפו את הקבצים הבאים:

1. `Queue.h`
2. `Queue.cpp`

עשו להם **commit** עם תיאור אינדיקטיבי, ודחפו אותם ל-Repo המרוחק בעזרת **push**.

חלק 2 – מימוש מחסנית

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.
במטלה זו אנחנו נממש מחסנית **באמצעות רשימה מקושרת**.

סעיף א – 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, לפני שאתם ממשיכים לסעיף הבא.

משימת גיט

- **הוסיפו** את השינויים שעשיתם בסעיף א'
- עשו להם **commit** עם תיאור אינדיקטיבי
- דחפו אותם ל-Repo המרוחק בעזרת **push**.

סעיף ב – מימוש מחסנית

כעת עליכם 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.

ניתן להוסיף פונקציות עזר כרצונכם.

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

צרו קובץ main.cpp ובדקו שהמחסנית שמימשתם עובדת כראוי.

משימת גיט

טוב נראה שכבר הבנתם...

בכל פעם שהגענו למצב שיש לנו קוד עובד – אנחנו עושים לו **add**, **commit** & **push**

מעכשיו סומכים עליכם שתעשו את זה כבר לבד **בסיום כל שלב בתרגיל** :)

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

סעיף ג – 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.

סעיף ד – הוספת קלט מהשתמש // משימת לימוד עצמי

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

עליכם 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 חלק 2

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

1. `LinkedList.h`
2. `LinkedList.cpp`
3. `Stack.h`
4. `Stack.cpp`
5. `Utils.cpp`

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 - **O(1)**

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

בהצלחה!