<u>מתרגל ממונה על התרגיל:</u> אמיר מן, amir.mann@campus.technion.ac.il

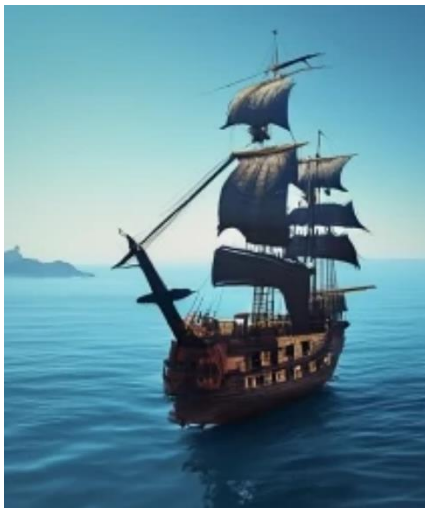
<u>תאריך ושעת הגשה:</u> 18/07/2024 בשעה 23:59

<u>אופן ההגשה:</u> בזוגות. אין להגיש ביחידים. (אלא באישור מתרגל אחראי של הקורס)

<u>הנחיות כלליות:</u>

- שאלות על התרגיל יש לפרסם באתר הפיאצה של הקורס תחת לשונית "hw-wet-1":
  - האתר: piazza.com/technion.ac.il/spring2024/234218
  - נא לקרוא את השאלות של סטודנטים אחרים לפני שמפרסמים שאלה חדשה, למקרה שנשאלה כבר.
- נא לקרוא את המסמך "נהלי הקורס" באתר הקורס. בנוסף, נא לקרוא בעיון את כל ההנחיות בסוף מסמך זה.
- בפורום הפיאצה ינוהל FAQ ובמידת הצורך יועלו תיקונים כ**הודעות נעוצות** (Pinned Notes). תיקונים אלו **מחייבים**.
- התרגיל מורכב משני חלקים: יבש ורטוב.
  - לאחר קריאת כלל הדרישות, מומלץ לתכנן תחילה את מבני הנתונים על נייר. דבר זה יכול לחסוך לכם זמן רב.
  - לפני שאתם ניגשים לקודד את פתרונכם, ודאו כי יש לכם פתרון העומד <u>בכל</u> דרישות הסיבוכיות בתרגיל. תרגיל שאינו עומד בדרישות הסיבוכיות יחשב כפסול.
  - את הפתרון שלכם מומלץ לחלק למחלקות שונות שאפשר לממש (ולבדוק!) בהדרגתיות.
  - המלצות לפתרון התרגיל נמצאות באתר הקורס תחת: "Programming Tips Session".
    - המלצות לתכנות במסמך זה <u>אינן</u> מחייבות, אך מומלץ להיעזר בהן.
  - חומר התרגיל הינו כל החומר שנלמד בהרצאות ובתרגולים עד אך לא כולל עצי דרגות.
- העתקת תרגילי בית רטובים תיבדק באמצעות תוכנת בדיקות אוטומטית, המזהה דמיון בין כל העבודות הקיימות במערכת, גם כאלו משנים קודמות. לא ניתן לערער על החלטת התוכנה. התוכנה אינה מבדילה בין מקור להעתק! אנא הימנעו מהסתכלות בקוד שאינו שלכם.
- בקשות להגשה מאוחרת יש להפנות למתרגל האחראי בלבד בכתובת: jonathan.gal@campus.technion.ac.il

CREATE TEMPLATE AVL TREE



הקדמה:

בעיית הפיראטים בעיים הקריביים החריפה מאד בשנים האחרונות ובשביל לעקוב אחר כולם ממשלת אנגליה נדרשת למבנה נתונים יעיל לניהול המידע עליהם.

אליזבת סוואן קיבלה את משימת הקמת התשתית, אך היא לא לקחה את הקורס במבני נתונים! היחלצו לעזרתה ועזרו לה לבנות את המערכת. הפיראטים מאורגנים בספינות, כך שכל פיראט, וספינה מזוהים על ידי ID יחודי, והמערכת תשמור בנוסף לכך גם מידע על סטטיסטיקות שונות ותאפשר לבצע ביעילות את השינויים שקורים בים.

לכל אורך המסמך בהתייחסות לסיבוכיות זמן ריצה ומקום הסימון n הוא מספר הפיראטים ו-m הוא מספר הספינות.

דרוש מבנה נתונים למימוש הפעולות הבאות:

oceans_t()

מאתחלת מבנה נתונים ריק. תחילה אין במערכת ספינות או פיראטים.

פרמטרים: אין

ערך החזרה: אין

סיבוכיות זמן: $O(1)$ במקרה הגרוע.

---

virtual ~oceans_t()

הפעולה משחררת את המבנה (כל הזיכרון אותו הקצאתם חייב להיות משוחרר).

פרמטרים: אין

ערך החזרה: אין

סיבוכיות זמן: $O(n+m)$ במקרה הגרוע.

---

key

StatusType add_ship(int shipId, int cannons)

ישנה ספינה חדשה בעל מזהה shipId. בהתחלה לספינה אין פיראטים בכלל, ויש לה cannons תותחים.

פרמטרים:

shipId	מזהה הספינה שצריך להוסיף.
cannons	מספר התותחים של הספינה.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם cannons < 0 או shipId <=0.
FAILURE	אם קיימת כבר ספינה עם מזהה shipId.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(\log m)$ במקרה הגרוע, כאשר m הוא מספר הספינות במערכת.

AVL tree is binary so searching is only logm - no need to hold array

each ship node holds pirate AVL tree

StatusType remove_ship(int shipId)

הספינה בעלת מזהה shipId אינה בשימוש יותר, ולכן צריך להוציאה מהמערכת.
אם קיימים פיראטים על הספינה - לא ניתן למחוק אותה, היא נשארת במערכת והפעולה נכשלת.

פרמטרים:

shipId	מזהה הספינה.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם shipId<=0.
FAILURE	אם אין ספינה עם מזהה shipId או שיש פיראטים בספינה זו.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(\log m)$ במקרה הגרוע, כאשר m הוא מספר הספינות במערכת.

StatusType add_pirate(int pirateId , int shipId, int treasure)

פיראט בעל מזהה ייחודי pirateId עולה על הספינה עבור ספינה shipId. לפיראט אוצר עם treasure מטבעות, שיכול להיות גם שלילי.

פרמטרים:

pirateId	מזהה הפיראט שצריך להוסיף.
shipId	מזהה הספינה של הפיראט.
treasure	כמות המטבעות באוצר של הפיראט.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם pirateId <=0, או shipId<=0
FAILURE	אם קיים כבר פיראט עם מזהה pirateId או שהספינה עם המזהה shipId לא קיימת.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(\log n + \log m)$ במקרה הגרוע, כאשר n הוא מספר הפיראטים במערכת, ו-m הוא מספר הספינות במערכת.

logm to find ship + log n to find pirate (both AVL trees)

rotation to fix tree is O(1)

StatusType remove_pirate(int pirateId)

הפיראט בעל מזהה pirateId קיבל מחלת ים והחליט לחזור לחוף, ולכן צריך להוציאו ממבנה הנתונים.

פרמטרים:

pirateId	מזהה הפיראט.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם pirateId <=0.
FAILURE	אם אין פיראט עם מזהה pirateId.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(\log n)$ במקרה הגרוע, כאשר n הוא מספר הפיראטים במערכת.

StatusType treason(int sourceShipId, int destShipId)

הפיראט שהיה הכי הרבה זמן (ברציפות) בספינה עם מזהה sourceShipId בוגד ועובר לספינה destShipId.

דוגמה:

make a queue of keys for the pirates. Then remove the first in

אם ישנן שתי ספינות 1 ו2.

לספינה 1 הגיעו פיראטים לפי הסדר הבא: 3 ולאחריו 4, ולאחריו 5. מצב המערכת:

1:[3, 4, 5] 2:[]

בגידה מספינה 1 לספינה 2 תגרום למצב מערכת:

1:[4, 5] 2:[3]

בגידה מספינה 2 לספינה 1 תגרום למצב מערכת:

1:[4, 5, 3] 2:[]

בגידה מספינה 1 לספינה 2 תגרום למצב מערכת:

1:[5, 3] 2:[4]

<u>פרמטרים</u>:

sourceShipId	מזהה הספינה ממנה בוגד פיראט.
destShipId	מזהה הספינה אליה יעבור הפיראט הבוגד.

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם sourceShipId<=0, destShipId<=0.
FAILURE	אם אין ספינות במזהה sourceShipId או destShipId במערכת, או שהספינה עם המזהה sourceShipId ריקה.
SUCCESS	במקרה של הצלחה.

<u>סיבוכיות זמן</u>: $O(\log m + \log n)$ במקרה הגרוע, כאשר m הוא מספר הספינות במערכת וn הוא מספר הפיראטים במערכת.

StatusType update_pirate_treasure(int pirateId , int change)

בעקבות תוצאות הימורים, הפיראט בעל מזהה pirateId משנה את כמות המטבעות באוצר שלו ב-change. תשנו את כמות המטבעות באוצר של הפיראט במערכת בchange. שימו לב, פיראט יכול להיות בחובות.

<u>פרמטרים</u>:

pirateId	מזהה הפיראט.
change	השינוי בכמות המטבעות באוצר של הפיראט.

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם pirateId<=0.
FAILURE	אם אין פיראט עם מזהה pirateId.
SUCCESS	במקרה של הצלחה.

find pirate in pirate AVL (O(logn)) + change val (O(1))

<u>סיבוכיות זמן</u>: $O(\log n)$ במקרה הגרוע, כאשר n הוא מספר הפיראטים במערכת.

---

output_t < int > get_treasure(int pirateId)

יש להחזיר את כמות המטבעות באוצר של הפיראט במזהה pirateId.

<u>פרמטרים</u>:

pirateId מזהה הפיראט.

<u>ערך החזרה</u>: כמות המטבעות באוצר של הפיראט, ובנוסף סטטוס:

ALLOCATION_ERROR במקרה של בעיה בהקצאה/שחרור זיכרון.

INVALID_INPUT אם pirateId <=0.

FAILURE אם אין פיראט עם מזהה pirateId.

SUCCESS במקרה של הצלחה.

<u>סיבוכיות זמן</u>: $O(\log n)$ במקרה הגרוע, כאשר n הוא מספר הפיראטים במערכת.

---

output_t < int > get_cannons(int shipId)

יש להחזיר את כמות התותחים שעל הספינה במזהה shipId.

<u>פרמטרים</u>:

shipId מזהה הספינה.

<u>ערך החזרה</u>: מספר התותחים הכולל שעל הספינה, ובנוסף סטטוס:

ALLOCATION_ERROR במקרה של בעיה בהקצאה/שחרור זיכרון.

INVALID_INPUT אם shipId <=0.

FAILURE אם אין ספינה עם מזהה shipId.

SUCCESS במקרה של הצלחה.

<u>סיבוכיות זמן</u>: $O(\log m)$ במקרה הגרוע, כאשר m הוא מספר הספינות במערכת.

---

output_t < int > get_richest_pirate(int shipId)

יש להחזיר את מזהה הפיראט העשיר ביותר (בעל מספר מטבעות מקסימלי באוצר שלו) מהספינה במזהה shipId.

שימו לב כי אין להחזיר את מספר המטבעות אלה את **מזהה הפיראט**. במידה וישנם מספר פיראטים בעלי אותה כמות מטבעות, המקסימלית על גבי הספינה, יש להחזיר את בעל המזהה המקסימלי מבינהם.

<u>פרמטרים</u>:

shipId מזהה הספינה.

<u>ערך החזרה</u>: מזהה הפיראט בעל מספר המטבעות הגדול ביותר באוצרו על הספינה, ובנוסף סטטוס:

ALLOCATION_ERROR במקרה של בעיה בהקצאה/שחרור זיכרון.

INVALID_INPUT אם shipId <=0.

FAILURE אם אין ספינה עם מזהה shipId או שאין פיראטים בספינה shipId.

SUCCESS במקרה של הצלחה.

<u>סיבוכיות זמן</u>: $O(\log m)$ במקרה הגרוע, כאשר m הוא מספר הספינות במערכת.

StatusType ships_battle(int shipId1, int shipId2)

שתי ספינות פיראטים נלחמות אחת בשניה, הספינה המנצחת היא זו שיש לה יותר תותחים וגם פיראטים לאייש אותם. פורמלית עם $c1, p1$ כמות הפיראטים והתותחים בספינה 1 וגם $c2, p2$ כמות הפיראטים והתותחים בספינה 2, אם $\min(c1, p1) > \min(c2, p2)$ אז ספינה 1 תנצח,
אם $\min(c1, p1) < \min(c2, p2)$ אז ספינה 2 תנצח,
אחרת הקרב נגמר בתיקו ודבר לא משתנה.
הספינה המנצחת שודדת מהספינה המפסידה, כך שכל פיראט בספינה המנצחת לוקח מטבע אחד מכל פיראט בספינה המפסידה.

דוגמה:

מצב מערכת התחלתי:
ספינה 1 - 3 תותחים
פיראט 3 – בעל 5 מטבעות באוצר שלו
פיראט 4 – בעל 4 מטבעות באוצר שלו
פיראט 5 – בעל 0 מטבעות באוצר שלו
פיראט 6 – בעל 2 מטבעות באוצר שלו
ספינה 2 - 12 תותחים
פיראט 7 – בעל 2 מטבעות באוצר שלו
פיראט 8 – בעל 10 מטבעות באוצר שלו

יש קרב בין ספינה 1 לספינה 2.
$\min(3, 4) > \min(12, 2)$ ולכן ספינה 1 מנצחת.
לכן מצב המערכת הופך להיות:
ספינה 1 - 3 תותחים
פיראט 3 – בעל 7 מטבעות באוצר שלו
פיראט 4 – בעל 6 מטבעות באוצר שלו
פיראט 5 – בעל 2 מטבעות באוצר שלו
פיראט 6 – בעל 4 מטבעות באוצר שלו
ספינה 2 - 12 תותחים
פיראט 7 – בעל 2- מטבעות באוצר שלו
פיראט 8 – בעל 6 מטבעות באוצר שלו

<u>פרמטרים</u>:

shipId1	מזהה הספינה הראשונה.
shipId2	מזהה הספינה השנייה.

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם shipId1<=0, shipId2<=0, shipId2==shipId1
FAILURE	אם אין ספינה עם מזהה shipId1 או shipId2.
SUCCESS	במקרה של הצלחה.

<u>סיבוכיות זמן</u>: $O(\log m)$ במקרה הגרוע, כאשר m הוא מספר הספינות.

<u>סיבוכיות מקום</u>**:**

סיבוכיות המקום הדרושה עבור מבנה הנתונים היא $O(n+m)$ במקרה הגרוע, כאשר n הוא מספר הפיראטים, וm הוא מספר הספינות. כלומר בכל רגע בזמן הריצה, צריכת המקום של מבנה הנתונים תהיה לינארית בסכום מספרי הפיראטים והספינות במערכת.

סיבוכיות המקום הנדרשת עבור כל פעולה (כלומר, זיכרון "העזר" שכל פעולה משתמשת בו) אינה מצוינת לכל פעולה לחוד, אך אסור לעבור את סיבוכיות המקום הדרושה שמוגדרת לכל המבנה.

<u>ערכי החזרה של הפונקציות:</u>

כל אחת מהפונקציות מחזירה ערך מטיפוס StatusType שייקבע לפי הכלל הבא:

- תחילה, יוחזר INVALID_INPUT אם הקלט אינו תקין.
- אם לא הוחזר INVALID_INPUT:
  - בכל שלב בפונקציה, אם קרתה שגיאת הקצאה/שחרור יש להחזיר ALLOCATION_ERROR. מצב זה אינו צפוי אלא באחד משני מקרים (לרוב): באמת השתמשתם בקלט גדול מאוד ולכן המבנה ניצל את כל הזיכרון במערכת, או שיש זליגת זיכרון בקוד.
  - אם קרתה שגיאה אחרת, כפי שמצוין בכל פונקציה, יש להחזיר מיד FAILURE <u>מבלי</u> לשנות את מבנה הנתונים.
- אחרת, יוחזר SUCCESS.

חלק מהפונקציות צריכות להחזיר בנוסף עוד פרמטר (לרוב int), לכן הן מחזירות אובייקט מטיפוס output_t<T>.

אובייקט זה מכיל שני שדות: הסטטוס (__status) ושדה נוסף (__ans) מסוג T.

במקרה של הצלחה (SUCCESS), השדה הנוסף יכיל את ערך החזרה, והסטטוס יכיל את SUCCESS. בכל מקרה אחר, הסטטוס יכיל את סוג השגיאה והשדה הנוסף לא מעניין.

שני הטיפוסים (output_t<T>,StatusType) ממומשים כבר בקובץ "wet1util.h" שניתן לכם כחלק מהתרגיל.

הנחיות:

חלק יבש:

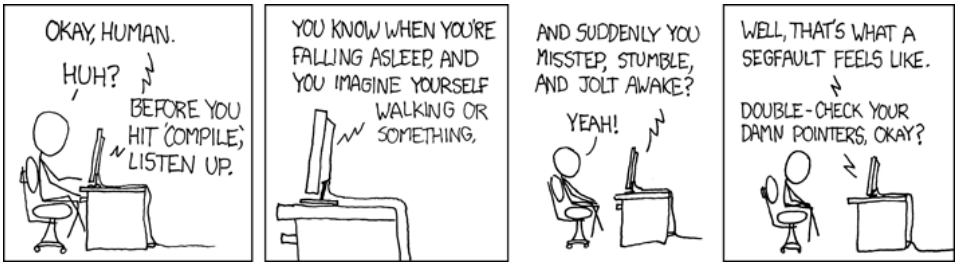
- החלק היבש הווה חלק מהציון על התרגיל כפי שמצוין בנהלי הקורס.
- לפני מימוש הפעולות בקוד יש לתכנן היטב את מבני הנתונים והאלגוריתמים ולוודא כי באפשרותכם לממש את הפעולות בדרישות הזמן והזיכרון שלעיל.
- החלק היבש חייב להיות מוקלד.
- הגשת החלק הרטוב מהווה תנאי הכרחי לקבלת ציון על החלק היבש, כלומר, הגשה בה יתקבל אך ורק חלק יבש תגרור ציון 0 על התרגיל כולו.
- יש להכין מסמך הכולל תיאור של מבני הנתונים והאלגוריתמים בהם השתמשתם בצירוף הוכחת סיבוכיות הזמן והמקום שלהם. חלק זה עומד בפני עצמו וצריך להיות מובן לקורא גם לפני העיון בקוד. אין צורך לתאר את הקוד ברמת המשתנים, הפונקציות והמחלקות, אלא ברמה העקרונית. חלק יבש זה לא תיעוד קוד.
- ראשית הציגו את מבני הנתונים בהם השתמשתם. רצוי ומומלץ להיעזר בציור.
- לאחר מכן הסבירו כיצד מימשתם כל אחת מהפעולות הנדרשות. הוכיחו את דרישות סיבוכיות הזמן של כל פעולה תוך כדי התייחסות לשינויים שהפעולות גורמות במבני הנתונים.
- הוכיחו שמבנה הנתונים וכל הפעולות עומדים בדרישת סיבוכיות המקום.
- החסמים הנתונים בתרגיל הם לא בהכרח הדוקים ולכן יכול להיות שקיים פתרון בסיבוכיות טובה יותר. מספיק להוכיח את החסמים הדרושים בתרגיל.
- רמת פירוט: יש להסביר את כל הפרטים שאינם טריוויאליים ושחשובים לצורך מימוש הפעולות ועמידה בדרישות הסיבוכיות. אין לדון בפרטים טריוויאליים (הפעילו את שיקול דעתכם בקשר לזה, ושאלו את האחראי על התרגיל אם אינכם בטוחים). אין לצטט קטעים מהקוד כתחליף להסבר. אין צורך לפרט אלגוריתמים שנלמדו בכתה. כמו כן, אין צורך להוכיח תוצאות ידועות שנלמדו בכתה, אלא מספיק לציין בבירור לאיזו תוצאה אתם מתכוונים. אין (וגם אין צורך) להשתמש בתוצאות של עצי דרגות והלאה.
- **על חלק זה לא לחרוג מ-8 עמודים.**
- והכי חשוב **keep it simple!**

חלק רטוב:

- מומלץ לממש תחילה את מבני הנתונים בצורה הכללית ביותר ורק אז לממש את הפונקציות הנדרשות בתרגיל.
- אנו ממליצים בחום על מימוש **Object Oriented**, ב**++C**, מימוש כזה יאפשר לכם להגיע לפתרון פשוט וקצר יותר לפונקציות אותן עליכם לממש ויאפשר לכם להכליל בקלות את מבני הנתונים שלכם (זכרו שיש תרגיל רטוב נוסף בהמשך הסמסטר).
- על הקוד להתקמפל על csl3 באופן הבא:

g++ -std=c++11 -DNDEBUG -Wall *.cpp

עליכם מוטלת האחריות לוודא קומפילציה של התכנית ב+ + g. אם בחרתם לעבוד בקומפיילר אחר, מומלץ לקמפל ב+ + g מידי פעם במהלך העבודה.



הערות נוספות:

- חתימות הפונקציות שעליכם לממש ומספר הגדרות נמצאים בקובץ pirates24b1.h.
- קראו היטב את הקובץ הנ"ל, לפני תחילת העבודה.
- אין לשנות את הקבצים main24b1.cpp ו-wet1util.h אשר סופקו כחלק מהתרגיל, **ואין להגיש אותם**.
  - את שאר הקבצים ניתן לשנות.
  - תוכלו להוסיף קבצים נוספים כרצונכם, ולהגיש אותם.
  - העיקר הוא שהקוד שאתם מגישים יתקמפל עם הפקודה לעיל, כאשר מוסיפים לו את שני הקבצים main24b1.cpp ו-wet1util.h.
- עליכם לממש בעצמכם את כל מבני הנתונים (למשל **אין להשתמש במבנים של STL** ואין להוריד מבני נתונים מהאינטרנט). **כחלק מתהליך הבדיקה אנו נבצע בדיקה ידנית של הקוד ונוודא שאכן מימשתם את מבני הנתונים שבהם השתמשתם.**
- בפרט, אסור להשתמש ב-std::iterator, std::vector, std::pair, או כל אלגוריתם של STL.

- **ניתן** להשתמש במצביעים חכמים (Smart pointers כמו shared_ptr), בספריית math או בספריית exception.
- חשוב לוודא שאתם מקצים/משחררים זיכרון בצורה נכונה (מומלץ לוודא עם valgrind). לא חייבים לעבוד עם מצביעים חכמים, אך אם אתם מחליטים כן לעשות זאת, לוודא שאתם משתמשים בהם נכון. (תזכרו שהם לא פתרון קסם, למשל, כאשר יוצרים מעגל בהצבעות)
- שגיאות של ALLOCATION_ERROR בד"כ מעידות על זליגה בזיכרון.
- מצורפים לתרגיל קבצי קלט ופלט לדוגמא, ניתן להריץ את התוכנה על הקלט ולהשוות עם הפלט המצורף.
- **שימו לב:** התוכנית שלכם תיבדק על קלטים שונים מקבצי הדוגמא הנ"ל, שיהיו ארוכים ויכללו מקרי קצה שונים. לכן, מומלץ **מאוד** לייצר בעצמכם קבצי קלט, לבדוק את התוכנית עליהם, ולוודא שהיא מטפלת נכון בכל מקרה הקצה.

הגשה:

- **חלק יבש+ חלק רטוב**:

הגשת התרגיל הנה **אך ורק** אלקטרונית דרך אתר הקורס.
יש להגיש קובץ **ZIP** שמכיל את הדברים הבאים:

- בתיקייה הראשית:
  - קבצי ה-Source Files שלכם. **למעט הקבצים main24b1.cpp ו-wet1util.h**, שאסור לשנות.
  - קובץ PDF בשם dry.pdf אשר מכיל את הפתרון היבש. החלק היבש חייב להיות מוקלד.
  - קובץ submissions.txt, המכיל בשורה הראשונה את שם, תעודת הזהות וכתובת הדוא"ל של השותף הראשון ובשורה השנייה את שם, תעודת הזהות וכתובת הדוא"ל של השותף השני. לדוגמה:

John Doe 012345678 doe@cs.technion.ac.il
Henry Taub 123456789 taub@cs.technion.ac.il

- **שימו לב כי אתם מגישים את כל שלושת החלקים הנ"ל.**
- אין להשתמש בפורמט כיווץ אחר (לדוגמה RAR), מאחר ומערך הבדיקה האוטומטי אינו יודע לזהות פורמטים אחרים.
- **יש לוודא שכאשר נכנסים לקובץ הזיפ הקבצים מופיעים מיד בתוכו ולא בתוך תיקיה שבתוך קובץ הזיפ.** עבור הגשה שבה הקבצים יהיו בתוך תיקייה, הבדיקה האוטומטית לא תמצא את הקבצים ולא תוכל לקמפל ולהריץ את הקוד שלכם ולכן תיתן אוטומטית 0.
- לאחר שהגשתם, יש באפשרותכם לשנות את התוכנית ולהגיש שוב. ההגשה האחרונה היא הנחשבת.
- הגשה שלא תעמוד בקריטריונים הנ"ל תפסל ותקנס בנקודות!
  - אחרי שאתם מכינים את ההגשה בקובץ zip מומלץ מאוד לקחת אותה לשרת ולהריץ את הבדיקות שלכם עליה כדי לוודא שאתם מגישים את הקוד שהתכוונתם להגיש בדיוק (ושהוא מתקמפל).

דחיות ואיחורים בהגשה:

- דחיות בתרגיל הבית תינתנה אך ורק לפי תקנון הקורס.
- 5 נקודות יורדו על כל יום איחור בהגשה ללא אישור מראש. באפשרותכם להגיש תרגיל באיחור של עד 5 ימים ללא אישור. תרגיל שיוגש באיחור של יותר מ-5 ימים ללא אישור מראש יקבל 0.
- במקרה של איחור בהגשת התרגיל יש עדיין להגיש את התרגיל אלקטרונית דרך אתר הקורס.
- בקשות להגשה מאוחרת יש להפנות למתרגל האחראי בלבד בכתובת jonathan.gal@campus.technion.ac.il. לאחר קבלת אישור במייל על הבקשה, מספר הימים שאושרו לכם נשמר אצלנו. לכן, אין צורך לצרף להגשת התרגיל אישורים נוספים או את שער ההגשה באיחור.

בהצלחה!