מבוא לתכנות מערכות – 234124

תרגיל בית 1
סמסטר חורף 2025

תאריך פרסום: 28.11.2024
זמן אחרון להגשה: 11.12.2024 עד השעה 23:55
מתרגל אחראי לתרגיל: מוחמד-בראא אגבאריה

1 הערות כלליות

- תרגיל זה מהווה 4% מהציון הסופי בקורס.
- התרגיל להגשה בזוגות בלבד.
- כל ההודעות והעדכונים הנוגעים לתרגיל זה יפורסמו באתר הקורס ב-GR.
- מענה לשאלות בנוגע לתרגיל יינתן אך ורק בפורום התרגיל בפיאצה (קישור באתר הקורס) או בסדנאות. אין לשלוח דוא"ל לגבי התרגיל.
- שימו לב – לא תינתנה דחיות למועד הגשת התרגיל.
- על כל הקוד שאתם כותבים לעמוד בקונבנציות המפורטות בקובץ "קונבנציות לכתיבת קוד" הנמצא באתר הקורס. אי עמידה בקונבנציות עלולה לגרור הורדת ניקוד.
- קבצי התרגיל מסופקים לכם ב-GitHub.
- תיקונים למסמך התרגיל מסומנים בצהוב.

2 הקדמה

המטרות של תרגיל בית זה הן כתיבת תכנית ראשונה בשפת C++ והתנסות ב-Git.
מומלץ לפתור את התרגיל לפי הסדר – תחילה את החלק היבש ולאחר מכן את החלק הרטוב. לכל שאלה או עזרה שתצטרכו בתרגיל ניתן לכתוב לנו בפורום הקורס בפיאצה או להגיע לסדנאות לעזרה בשיעורי הבית (זמנים ומיקומים מפורסמים באתר הקורס) ונשמח לעזור.
לפני פרסום שאלה בפורום אנא בדקו אם כבר נענתה – מומלץ להיעזר בכלי החיפוש שהוצגו במצגת האדמיניסטרציה בתרגול הראשון.

3 חלק יבש

בחלק הזה נתרגל עבודה עם Git.
כפי שראינו בהרצאות ובתרגולים, Git הוא כלי המאפשר לנו לנהל את גרסאות הקוד שלנו ולעבוד עליו בשיתוף פעולה עם מתכנתים אחרים (ועוד הרבה פיצ'רים חשובים אחרים...). Git הוא הכלי לניהול תצורה הכי נפוץ היום בעולם והוא חלק בלתי נפרד מהעבודה בתור מהנדס תוכנה.
בנוסף, בקורס שלנו נעבוד עם אתר GitHub בתור ה-(Remote) Git Server שלנו. לפני תחילת העבודה על התרגיל על שני השותפים להירשם ל-GitHub, ההרשמה חינמית לחלוטין.
שימו לב: איפה שרשום (סעיף זה יבוצע על ידי אחד השותפים בלבד), הכוונה היא לביצוע, אך מומלצים ואף רצוי שתעברו על החלקים הללו ביחד.

3.1 יצירת Repository פרטי

(סעיף זה יבוצע על ידי אחד השותפים בלבד)
כנסו ל-GitHub וצרו Repo חדש על ידי לחיצה על כפתור "New" המופיע בצד שמאל למעלה באתר. תנו שם ל-Repo, למשל "Matam HW1", וסמנו שעל ה-Repo להיות Private. לחצו על "Create Repository" ותועברו לדף של ה-Repo החדש שיצרתם!
על מנת שגם השותף או השותפה שלכם יוכלו להשתמש ב-Repo, למשל לדחוף אליו Commits חדשים, עליכם להוסיף אותם בתור Collaborators. כנסו לעמוד "Settings" של ה-Repo ולחצו על "Collaborators" בתפריט בצד שמאל. כעת לחצו על "Add people" ורשמו בחיפוש את שם המשתמש או כתובת המייל של מי שתרצו להוסיף ל-Repo. לאחר שתאשרו את הבחירה, תישלח הזמנה למייל של ה-Collaborator שהוספתם, אותה הם יצטרכו לאשר על מנת לראות ולהשתמש ב-Repo.

שימו לב – בקורס שלנו, כל ה-Repos שלכם ב-GitHub נדרשים להיות פרטיים.

3.2 משיכת קבצי התרגיל

(סעיף זה יבוצע על ידי אחד השותפים בלבד)
לאחר שיצרתם Repo משלכם ב-GitHub, נרצה להעלות אליו את הקבצים המסופקים לתרגיל.
תחילה, משכו את הקבצים המסופקים לתרגיל הבית מה-Repo של סגל הקורס באמצעות פקודת clone, כפי למדנו בהרצאות ובתרגולים. כעת נוצר לכם עותק מקומי במחשב שלכם ל-Repo של הסגל. נרצה "לחבר" את ה-Repo שמשכתם ל-Repo ב-GitHub שיצרתם בסעיף הקודם.
כפי שראינו בהרצאה ובתרגול, לכל Repo מקומי במחשב שלכם מוגדר Remote אליו הוא מקושר. ה-Remote הוא למעשה עותק של ה-Repo המאוחסן בשרת Git (במקרה שלנו זהו GitHub). כשביצעתם Clone ל-Repo, ה-Remote אוטומטית הוגדר להיות ה-Repo ממנו ביצעתם Clone (במקרה זה ה-Repo של הסגל). תוכלו לראות את כל ה-Remotes המוגדרים ל-Repo ואת כתובתם באמצעות הרצת הפקודה:

```
git remote -v
```

הוסיפו Remote ל-Repo המקומי באמצעות הפקודה (החליפו את remote_name בשם לבחירתכם ואת url בכתובת ה-Repo שלכם ב-GitHub):

```
git remote add remote_name url
```

כעת תוכלו לדחוף את כל השינויים ל-Repo שלכם באמצעות הפקודה:

```
git push -u remote_name branch_name
```

למעשה הפקודה הזו מבקשת לדחוף את השינויים ב-Branch עליו אתם נמצאים ל-Remote הספציפי שבחרתם, וגרמת ל-Branch המקומי "לעקוב" אחרי ה-Branch ב-Remote שבחרתם. תצטרכו להריץ את הפקודה הזו בכל פעם שתרצו לדחוף Branch חדש ל-GitHub שלכם, בכל מקרה אחר ניתן להשתמש בפקודת push רגילה כפי שנלמדה בתרגולים.

שימו לב – בהרצת פקודת Git מה-Shell שדורשות הרשאות גישה ל-GitHub (למשל פקודת push), יתכן שתתבקשו להכניס שם משתמש וסיסמה. הכניסו את שם משתמש ה-GitHub שלכם. במקום סיסמה, יש להכניס Token אותו ניתן לייצר כאן. יש ללחוץ על "Generate new token", לבחור ב-"Generate new token (classic)" ובעמוד שנפתח לסמן את הרובריקה "repo" על מנת לקבל הרשאות מתאימות.
כדי ששם המשתמש וה-Token ישמרו במחשב ולא תצטרכו להכניס אותם בכל פעם, הריצו את הפקודה הבאה (לפני פקודת ה-push):

```
git config credential.helper store
```

לאחר הדחיפה הראשונה ל-Repo שלכם, תוכלו להתחיל לעבוד עליו כרגיל באמצעות הפקודות שנלמדו בתרגול ולא תצטרכו יותר את ה-Repo של הסגל (אלא במקרה של עדכונים בקבצי התרגיל).

3.3 ה-Commit הראשון שלי

(סעיף זה יבוצע על ידי שני השותפים, כל אחד בנפרד)
בקבצי התרגיל מסופק לכם קובץ בשם introduction.md, הקובץ הוא בפורמט שנקרא markdown, אם תכנסו לקובץ בגיטהאב של הסגל תוכלו לראות איך גיטהאב מציג קובץ מפורמט markdown. בסעיף זה תתבקשו לבצע שינויים לקובץ, כל אחד מהשותפים בנפרד, לשלב את השינויים שלכם באמצעות Git ולבסוף להגיש את הקובץ המכיל את השינויים שביצעתם.
בתחילת הקובץ תמצאו לינקים לשירים האהובים על המתרגלים בקורס (מוזמנים להאזין תוך כדי העבודה על התרגיל). אחרי השירים, תוכלו למצוא שני חלקים בהם תוכלו למלא מספר פרטים מעניינים על עצמכם.

עקבו אחר הצעדים הבאים:

1. פתחו את הקובץ introduction.md עם notepad או הפקודה nano שראיתם בתרגיל 0, או בעורך טקסט אחר לבחירתכם, בצעו את השינויים הרלוונטיים בקובץ ושימרו אותם. על כל אחד מהשותפים למלא את אחד מהחלקים בנפרד (שותף אחד את חלק 1 בקובץ והשני את חלק 2), למשל:

- First student's name: *Baraa Egbaria*
- Favorite singer: *The Lumineers*
- Favorite song: *Sleep on the Floor*
- Desirable grade in matam: *100*

2. צרו Commit חדש המכיל את השינויים שביצעתם. רשמו Commit Message מתאימה המפרטת את השינויים, כגון: "Update first student info".

   שימו לב – Best Practice לכתיבת Commit Message הוא שההודעה תהיה בלשון הווה ולא בלשון עבר (למשל "Fix a bug..." ולא "Fixed a bug").

3. דחפו את השינויים שביצעתם ל-Repo שלכם באמצעות פקודת push. בצעו pull עם rebase במידת הצורך.

3.4 הגשת החלק היבש

לאחר ששני השותפים דחפו את השינויים, תוכלו לראות ב-Repo שלכם את הקובץ introduction.md המעודכן והמלא. צרפו את הקובץ להגשה שלכם.

נרצה לצרף להגשה גם את ה-Log של ה-Repo שלכם. הריצו את הפקודה הבאה דרך ה-Shell:

```
git log -p
```

כפי שלמדנו בתרגול, פקודה זו מפרטת את כל ה-Commits שנעשו עד כה ב-Repo והדגל p- גורם להדפסה של כל השינויים עבור כל אחד מה-Commits.
צרפו להגשה צילום מסך של ה-Log שלכם בו ניתן לראות את ה-Commit של כל אחד מהשותפים עם השינויים המלאים של כל Commit. את הצילום צרפו להגשה בקובץ PDF בשם dry.pdf.

4 חלק רטוב

בראא, המתרגל האחראי בקורס, נפגש עם המתרגל רגב בקורס קריפטוגרפיה מודרנית (236506) שמועבר בפקולטה למדעי המחשב. הם החליטו לנצל את הידע שצברו בקורס ואת יכולות הפיתוח שלהם ב-C++, כמו גם להיעזר בכם הסטודנטים בקורס מת"מ, כדי להיכנס לתחום המטבעות הקריפטוגרפים.

Fun Fact – המימוש המקורי והנפוץ ביותר היום של Bitcoin נכתב בשפת C++, תוכלו לראות אותו כאן.

שימו לב – בחלק זה מומלץ להשתמש ב-Git ו-GitHub, כפי שלמדנו בהרצאות ובתרגולים ותרגלנו בחלק היבש. החל מתרגיל הבית הבא השימוש ב-Git ו-GitHub לכל חלקי התרגיל יהיה בגדר חובה.

4.1 Transaction

כל עסקה קריפטוגרפית (טרנזקציה) מורכבת משלושה אלמנטים בסיסיים – שולח, מקבל וסכום. לדוגמה, עסקה אפשרית היא העברה של 100 מטבעות (הסכום) מעדי (השולח) לבראא (המקבל).
בנוסף, לכל עסקה נגדיר מזהה יחודי. המזהים בהם נשתמש בתרגיל זה הם רצפים של 20 תווים הקסאדצימליים (ספרות או אותיות מ-a עד f).
מזהה של עסקה מתקבל על ידי הפעלה של פונקציית גיבוב (hash function) המקבלת מפתח ושתי מחרוזות.
בתרגיל זה המפתח הוא סכום העסקה והמחרוזות הן שם השולח ושם המקבל (בסדר הזה).
בחלק זה נגדיר ונממש את המודול Transaction המאפשר עבודה בסיסית עם עסקאות.

שימו לב – לשתי עסקאות שונות יכול להיות אותו מזהה.

4.1.1 ממשק המודול

בקבצים המסופקים תוכלו למצוא את הקובץ Transaction.h. בקובץ זה מוגדר ממשק המודול, המכיל את האלמנטים הבאים:

1. Transaction – מבנה המייצג עסקה ולו שדות המייצגים את סכום העסקה, השולח והמקבל.
2. TransactionDumpInfo – פונקציה מקבלת עסקה וקובץ פתוח, מדפיסה את פרטי העסקה לקובץ לפי הפורמט הבא:

Sender Name: <*sender_name*>
Receiver Name: <*receiver_name*>
Transaction Value: <*value*>

3. TransactionHashMessage – פונקציה המקבלת עסקה ומחזירה את המזהה הייחודי שלה.
4. TransactionVerifyHashedMessage – פונקציה המקבלת עסקה ומזהה. הפונקציה מחזירה true אם ורק אם המזהה שהתקבל הינו המזהה היחודי המתאים לעסקה.

שימו לב – אין לשנות את הממשק הנתון למודול זה.

4.1.2 מימוש המודול

עליכם לממש את פונקציות המודול בקובץ Transaction.cpp, אותו תצרפו להגשה שלכם. לשם קבלת מזהים של עסקות מסופקת לכם הפונקציה hash במודול בשם "Utilities" (בקבצים המסופקים Utilities.h ו-Utilities.cpp). אין צורך להבין איך הפונקציה hash עובדת.

ניתן להניח את ההנחות הבאות על קלט הפונקציות:

- הערכים בעסקה תמיד תקינים, כלומר המספרים תמיד חיוביים והשמות אינם ריקים.
- פונקציית hash תמיד מצליחה ומחזירה מזהה חוקי.

4.2 BlockChain

שימו לב: כדי לממש את הפונקציונליות של פונקציות 5 עד 8, נצטרך חומר שיילמד בתרגול 4 (שעדיין לא למדנו ברגע פרסום התרגיל).
כדי לנהל מספר עסקאות קריפטוגרפיות נהוג להשתמש במבנה נתונים הנקרא BlockChain. מבנה נתונים זה אינו מוגבל בכמות העסקאות שניתן לאחסן בו. כל פיסת מידע בשרשרת נקראת "בלוק" וכל בלוק מכיל מידע על עסקה או מספר עסקאות. ראש השרשרת (הבלוק הראשון) מכיל את העסקה האחרונה שבוצעה. בתרגיל זה, כל בלוק יכיל עסקה בודדת ואת הזמן והשעה בה התבצעה העסקה.
בחלק זה נגדיר ונממש את המודול BlockChain המאפשר עבודה עם רצפים של עסקאות.

שימו לב – מותר לכם להוסיף לממשק המודול פונקציות נוספות, אך הקפידו לא להוסיף אלמנטים מיותרים. כמו כן אסור לכם לשנות את הממשק הקיים (למשל לשנות חתימות של פונקציות).

4.2.1 ממשק המודול

בקבצים המסופקים תוכלו למצוא את הקובץ BlockChain.h. בקובץ זה מוגדר ממשק המודול, המכיל את האלמנטים הבאים:

1. BlockChain – מבנה המייצג שרשרת עסקאות. **עליכם להשלים את שדות המבנה**, בהתאם לצרכי הפתרון שלכם.

2. BlockChainGetSize – פונקציה המקבלת BlockChain ומחזירה את כמות העסקאות השמורות בו.

3. BlockChainPersonalBalance – פונקציה המקבלת BlockChain ושם של לקוח. הפונקציה מחזירה את המאזן של אותו לקוח ב-BlockChain, כלומר סך כל המטבעות אותם קיבל פחות סך כל המטבעות אותם נתן לאורך השרשרת.

4. BlockChainAppendTransaction – פונקציה המוסיפה עסקה חדשה לראש ה-BlockChain. לפונקציה זו שתי גרסאות:
   a. גרסה המקבלת כפרמטרים BlockChain, שמות של שולח ומקבל, כמות מטבעות וזמן ביצוע העסקה.
   b. גרסה המקבלת כפרמטרים BlockChain, עסקה (מבנה מסוג Transction) וזמן ביצוע העסקה.

5. BlockChainLoad - קוראת מקובץ מידע על עסקאות ומחזירה BlockChain המכיל את כל העסקאות. הקובץ מכיל את כל העסקאות בפורמט הבא:

<sender> <receiver> <value> <timestamp>
<sender> <receiver> <value> <timestamp>

הערות:

- העסקה הראשונה בקובץ היא ראש השרשרת, כלומר העסקה האחרונה שהתבצעה.
- ניתן להניח שתוכן הקובץ תקין, כלומר כל הנתונים נמצאים ומהטיפוס הנכון.
- ניתן להניח שהעסקאות בקובץ ממוינת לפי ה-timestamp.

6. BlockChainDump – פונקציה המקבלת BlockChain וקובץ פתוח. הפונקציה מדפיסה לקובץ את המידע על כל הבלוקים בשרשרת בפורמט הבא:

BlockChain Info:
1.
Sender Name: <*sender_name*>
Receiver Name: <*receiver_name*>
Transaction Value: <*value*>
Transaction timestamp: <*time*>
2.
Sender Name: <*sender_name*>
Receiver Name: <*receiver_name*>
Transaction Value: <*value*>
Transaction timestamp: <*time*>
...

שימו לב – הבלוק הראשון שמודפס הוא ראש השרשרת, כלומר העסקה האחרונה שבוצעה.

7. BlockChainDumpHashed – פונקציה המקבלת BlockChain וקובץ פתוח. הפונקציה מדפיסה לקובץ את המזהים היחודיים של כל עסקה בשרשרת בפורמט הבא:

<transaction_hash>
<transaction_hash>
...

שימו לב – המזהה הראשון שמודפס הוא המזהה של ראש השרשרת, כלומר של העסקה האחרונה שבוצעה. בנוסף, אין להדפיס ירידת שורה בסוף הקובץ (הקובץ מסתיים מיד בסיום המזהה האחרון).

8. BlockChainVerifyFile – פונקציה המקבלת BlockChain וקובץ פתוח המכיל מזהים של עסקאות. הפונקציה מחזירה true אם ורק אם המזהים מתאימים לכל העסקאות ב-BlockChain לפי הסדר (המזהה הראשון בקובץ מתאים לראש ה-BlockChain). כלומר, אם נקרא ל-BlockChainDumpHashed עבור השרשרת שהתקבלה נקבל את אותם מזהים באותו הסדר.

9. BlockChainCompress – פונקציה המקבלת BlockChain, ודוחסת אותו. כל הבלוקים עם אותו שולח ואותו מקבל שמופיעים ברצף בשרשרת מוחלפים בבלוק יחיד המכיל את אותם שולח ומקבל, את סכום המטבעות בכל הבלוקים המוחלפים ואת זמן העסקה של הבלוק האחרון שהוחלף.

10. BlockChainTransform – פונקציה המקבלת BlockChain ואופרציה (מצביע לפונקציה). הפונקציה מעדכנת את הערך של כל העסקאות בשרשרת בהתאם להפעלת האופרציה על כמות המטבעות בעסקה.
לדוגמה – אם התקבלה אופרציה המקבלת פרמטר ומחזירה את המכפלה שלו ב-2, כמות המטבעות בכל אחת מהעסקאות תוכפל ב-2.

```
unsigned int timesTwo(unsigned int x) {
    return x * 2;
}

int main() {
    BlockChain blockChain;
    /* some code, assume blockChain holds 1->3->7 */
    BlockChainTransform(blockChain, timesTwo);
    /* blockChain now holds 2->6->14 */
}
```

4.2.2 מימוש המודול

עליכם לממש את פונקציות המודול בקובץ BlockChain.cpp, אותו תצרפו להגשה שלכם.
הניחו שהפרמטרים המועברים לפונקציות השונות תמיד תקינים. כמו כן, הניחו שקבצים המתקבלים בשורת הפקודה קיימים ומכילים מידע בפורמט המתאים.

4.3 התכנית mtm_blockchain

בחלק זה נממש תוכנית בשם mtm_blockchain, המשתמשת במודול ה-BlockChains. ממשו את התכנית בקובץ main.cpp והוסיפו את הקובץ להגשה שלכם.
התוכנית תקבל דרך ה-Shell שלושה ארגומנטים – שם פקודה, ושני שמות של קבצים:

```
./mtm_blockchain op source target
```

op הינו שם הפקודה המעיד על סוג הפעולה שהתכנית תבצע. שם הפקודה יכול להיות אחד מארבעת הבאים:

1. format - קוראת מידע על BlockChain מהקובץ source ומדפיסה את המידע על כל העסקאות לקובץ target, לפי פורמט ההדפסה של BlockChainDump.

2. hash – קוראת מידע על BlockChain מהקובץ source ומדפיסה את המזהים של כל העסקאות לקובץ target, לפי פורמט ההדפסה של BlockChainDumpHashed.

3. compress - קוראת מידע על BlockChain מהקובץ source, דוחסת אותו (באמצעות BlockChainCompress) ומדפיסה את המידע על כל העסקאות לקובץ target, לפי פורמט ההדפסה של BlockChainDump.

4. verify - קוראת מידע על BlockChain מהקובץ source וקוראת את המזהים של כל העסקאות מקובץ ה-target. התכנית בודקת אם המזהים ב-target אכן שייכים לעסקאות מהקובץ source (לפי הסדר). התכנית תדפיס לפלט הסטנדרטי את ההודעה הבאה (עם passed במידה והמזהים מתאימים ועם failed אחרת):

Verification <failed/passed>

במידה ולא מתקבלים מספיק ארגומנטים או מתקבלים יותר מדי ארגומנטים, תודפס הודעת השגיאה הבאה והתכנית תצא:

Usage: ./mtm_blockchain <op> <source> <target>

שימו לב – עבור ההדפסות של הודעות השגיאה וההצלחה מסופקות לכן פונקציות מתאימות במודול Utilities.

4.3.1 הידור ובדיקה

את התכנית יש להדר באמצעות הפקודה הבא:

```
g++ -DNDEBUG -std=c++17 -Wall -pedantic-errors -Werror -o mtm_blockchain *.cpp
```

כמו כן עם קבצי התרגיל מסופקים לכם מספר מקרי בדיקה. תוכלו לבדוק אותם באמצעות הפקודות הבאות:

```
./mtm_blockchain hash tests/hash.source tests/hash.target.out

./mtm_blockchain compress tests/compress.source tests/compress.target.out

./mtm_blockchain format tests/format.source tests/format.target.out

./mtm_blockchain verify tests/verify.source tests/verify.target > verify.out
```

5 הגשה ובדיקה

5.1 הגשה

עליכם להגיש את הקבצים הבאים:

- main.cpp
- BlockChain.cpp
- BlockChain.h
- Transaction.cpp
- introduction.md
- dry.pdf

אין להגיש את הקבצים Utilities.h, Utilities.cpp או Transaction.h.

את ההגשה יש לבצע במערכת ה-Gradescope, אליה תוכלו למצוא קישור באתר הקורס. תוכלו להגיש בשני אופנים שונים, הגשת zip או הגשה באמצעות GitHub, עוד על כך בסעיפים הבאים.

שימו לב – על אחד השותפים (בלבד) להגיש את פתרון התרגיל במערכת. לאחר ההגשה ניתן להוסיף את השותף השני להגשה באמצעות לחיצה על כפתור "Group Members" ב-Gradescope.

הערות:

- ניתן להגיש מספר פעמים את התרגיל, ההגשה האחרונה (בלבד) היא זו שתיבדק ותקבל ציון.
- וודאו שהקבצים אותם אתם מגישים (בפרט קבצי PDF) ניתנים לפתיחה ולצפיה.

5.1.1 הגשת zip

ניתן להגיש את הפתרון שלכם בתור קובץ zip כפי שהגשתם את הפתרון לתרגיל בית 0. על הקובץ להיות מכווץ כ-zip (לא rar או כל דבר אחר) כאשר קבצי ההגשה נמצאים בתיקיה הראשית בקובץ ה-zip.

5.1.2 הגשה באמצעות GitHub

בתרגיל זה, ניתן להגיש את הפתרון שלכם במערכת ה-Gradescope גם באמצעות GitHub. תוכלו להגיש את ה-Repo שיצרתם בחלק היבש (בהנחה והוספתם אליו גם את שאר פתרון התרגיל). לשם כך בדף הגשת התרגיל בחרו באפשרות ההגשה באמצעות GitHub, תנו ל-Gradescope הרשאות גישה לחשבון ה-GitHub שלכם ובחרו את ה-Repo אותו תרצו להגיש.

שימו לב – בתרגיל זה מומלץ להגיש את הפתרון באמצעות GitHub, על מנת שתתנסו בעבודה עם Git ובהגשה מסוג זה. החל מהתרגילים הבאים הגשה באמצעות GitHub תהיה בגדר חובה.

5.2 בדיקה אוטומטית

מיד עם ההגשה שלכם במערכת ה-Gradescope, יתבצעו מספר בדיקות אוטומטיות על קובץ ההגשה:

1. בדיקות שפיות – בודקות שכל הקבצים הדרושים נמצאים ב-zip המוגש ושהקוד שלכם מתקמפל כראוי. בדיקות אלו נועדו לוודא שפורמט ההגשה שלכם תקין.
2. טסטים אוטומטיים – הבדיקה האוטומטית מריצה את התכניות שהגשתם עם הקלטים שסופקו לכם בקבצי התרגיל. הטסטים האוטומטיים בודקים שריצת התכנית הסתיימה בהצלחה ללא שגיאות זכרון ונתנה פלט נכון.

בדיקות אלו נועדו עבורכם לוודא שההגשה שלכם תקינה והקוד עובד כראוי, קראו היטב את הפלט והמשוב מהבדיקות!

על מנת להשוות תכנים של קבצים בבדיקה האוטומטית, נשתמש בפקודת ה-diff הבאה (כאשר file1 ו-file2 הם הקבצים אותם נרצה להשוות:

```
diff --strip-trailing-cr -B -Z file1 file2
```

<u>שימו לב</u> – בנוסף לבדיקות שסופקו לכם, ההגשה הולכת לעבור בדיקות אוטומטיות נוספות לפיהן יקבע הציון על התרגיל, <u>כולל בדיקת פונקציות שלא נבדקו בטסטים שסופקו לכם</u>. אנו מצפים שתבדקו בעצמכם את התרגיל שלכם מעבר לבדיקות המסופקות, האחריות על נכונות ההגשה היא שלכם בלבד.

5.3 תחרות מימז

בתרגיל זה מתקיימת תחרות מימז (Memes).
כדי להשתתף בתחרות, עליכם להוסיף מימ לתחילת הקובץ dry.pdf. הזוג שיזכה בתחרות יקבל בונוס של 5 נקודות לציון התרגיל (לכל אחד מהשותפים).

חוקי התחרות:

1. מותר לכם להגיש לתחרות מימ אחד <u>בלבד</u>.
2. נושא המימ חייב להיות קשור לחומר הקורס.
3. המימ הזוכה יפורסם בתרגול לאחר לפרסום הציונים.

בהצלחה!

بالنجاح!

GOOD LUCK!