

# תרגול 2

## רשתות חברתיות

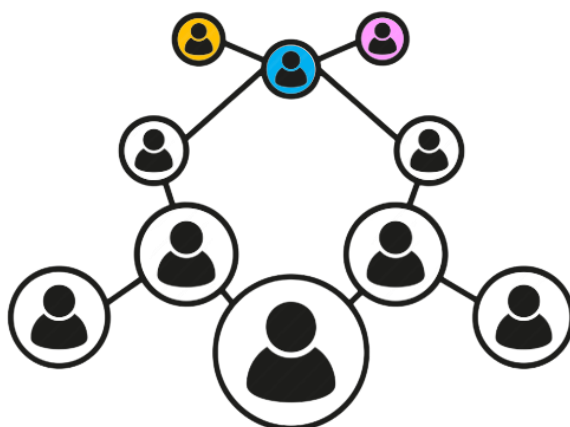
### רקע

שיעור 2 היה עמוס מאוד, בעיקר בגלל שלמדנו בו גישת תכנות חדשה – תכנות מונחה עצמים. בגישת OOP (Object Oriented Programming) אנחנו מפצלים את התוכנית שלנו לישויות עצמאיות שלהן אנחנו קוראים מחלקות, חלק מהמחלקות מבודדות, וחלק משתמשות במחלקות אחרות. כל מחלקה עושה תפקיד מסוים, וביחד כל החלקים מרכיבים תוכנה או מערכת שלמה.

אחד היתרונות שיש לגישת OOP הוא שאפשר לבנות מערכות מורכבות בצורה הדרגתית, חלק אחר חלק. אבל כדי לשלוט בגישת התכנות החדשה צריך לתרגל, לכן לאורך הסמסטר נעבוד על פיה.

### מטרה

בתרגיל נבנה רשת חברתית צנועה שבה משתמשים יוכלו להתחבר לפרופיל שלהם באמצעות מכשירים שונים, לכתוב פוסטים, ולהוסיף חברים.



אלה השלבים שנעבור:

למידה עצמית האופרטור &	למידה עצמית מחרוזות (std::string)	כתיבת מחלקת Device	כתיבת מחלקת User	כתיבת מחלקות Profile ו Page	כתיבת מחלקת Social Network
<ul style="list-style-type: none"> <li>למידה עצמית על נושא reference הגשת תשובות לשאלות תאורטיות</li> </ul>	<ul style="list-style-type: none"> <li>למידה עצמית על נושא מחרוזות</li> </ul>	<ul style="list-style-type: none"> <li>כתיבת הגדרות מימוש המחלקה</li> </ul>	<ul style="list-style-type: none"> <li>כתיבת הגדרות מימוש המחלקה שימוש ברשימה מקושרת</li> </ul>	<ul style="list-style-type: none"> <li>כתיבת הגדרות מימוש המחלקות שימוש ברשימה מקושרת שלב בונוס</li> </ul>	<ul style="list-style-type: none"> <li>כתיבת הגדרות מימוש המחלקה שימוש ברשימה מקושרת</li> </ul>

נתרגל מיומנויות חשובות:

- נתרגל שימוש במשתנים מסוג reference.
- נתנסה בכתיבת מחלקות ושימוש נכון בבימוס (encapsulation).
- נשתמש במחלקה `std::string`.
- נעבוד עם רשימות מקושרות.

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

כדאי לקרוא גם [דגשים לתכנות נכון](#).



"PRACTICE MAKES PERFECT"

**בהצלחה יא אלופות ואלופים!**

## שלב 1: לימוד עצמי – האופרטור &

שלב 6 מחלקת Social Network	שלב 5 מחלקות Profile ו- Page	שלב 4 מחלקת User	שלב 3 מחלקת Device	שלב 2 מחרוזות	שלב 1 האופרטור &
-------------------------------	---------------------------------	---------------------	-----------------------	------------------	---------------------

### למידה עצמית

קראו ב**מאמר הבא** על המושג reference וענו על השאלות הבאות:

1. מהו reference וכיצד מגדירים reference למשתנה?
2. מנו שני יתרונות בהעברת משתנים לפונקציה by reference על פני שיטות אחרות.
3. מה הם ההבדלים בין pointer לreference? מדוע reference נחשב ל"בטוח" יותר?
4. נתונה הפונקציה ונתון משתנה y מסוג int:

```
void square(int x, int& result)
{
    result = x * x;
}
```

האם הקריאות הבאות תקינות? אם לא, הסבירו מה הבעיה.

- א. `square(3, y);`
- ב. `square(3, &y);`
- ג. `square(3, 6);`

5. מה הבעיה בכל אחת מהפונקציות הבאות:

a.

```
int& getLocalVar()
{
    int x = 10;
    return x;
}
```

b.

```
int& getDynamicVar()
{
    int *x = new int(10);
    return *x;
}
```

סיכום חלק 1 – עליכם/ן להעלות לגיט את הקבצים הבאים: **answers.pdf**

## שלב 2: מחרוזות

שלב 6 מחלקת Social Network	שלב 5 מחלקות Profile - I Page	שלב 4 מחלקת User	שלב 3 מחלקת Device	שלב 2 מחרוזות	שלב 1 האופרטור &
-------------------------------	----------------------------------	---------------------	-----------------------	------------------	---------------------

### למידה עצמית - מחלקת `std::string`

בחלק הזה נביר מחלקה מאוד שימושית אשר מומשה עבורנו בספריית `std` - הספרייה הסטנדרטית של C++.

לאורך השנה, וגם בתוכניות שנכתוב בהמשך, נראה שאחד הדברים הנפוצים שחוזרים הוא שימוש במחרוזות = רצף של תווים.

עד עכשיו עבדנו בשפת C עם מערך תווים (`char [ ]`), או עם מצביעים שאליהם הקצינו זיכרון (`char*`). יכול להיות שחלקנו אפילו השתמש בספריות חיצוניות שהכילו פונקציות שימושיות שעובדות עם הסוגים האלו כמו לדוגמא `strcmp()`, `strtok()`, `strcat()`.

עכשיו לאחר שלמדנו על מחלקות ב-C++, נוכל להביר מחלקה שימושית אשר נועדה לייצג רצף תווים – `std::string`.

אובייקט מסוג `std::string` נבנה באמצעות רצף תווים.

הנה כמה דוגמאות ליצירת מופע של המחלקה `std::string`

```
std::string my_string1 = "";
std::string my_string2 = "lalalala";
std::string my_string3 = "123 and 345";
std::string my_string4 = "#!@%$^&*()";

const char* c_str = "bla bla";
std::string my_string = c_str; // mystring = "bla bla"
```

באובייקט של `std::string` קיימים אופרטורים אשר הופכים אותו להיות דומה מאוד למערך התווים אליו אנחנו רגילים, לדוגמא האופרטור `[ ]` (סוגריים מרובעים) מאפשר לגשת לתו במחרוזת כמו שאנו ניגשים לתא במערך:

```
std::string my_string3 = "123 and 456";
char c1 = my_string3[5]; // c1 = 'n'
char c2 = my_string3[2]; // c2 = '3'
char c3 = my_string3[10]; // c2 = '5'
char c4 = my_string3[12]; // run time error!!!
```

```
std::string s1 = "Live";
s1[1] = 'o';
```

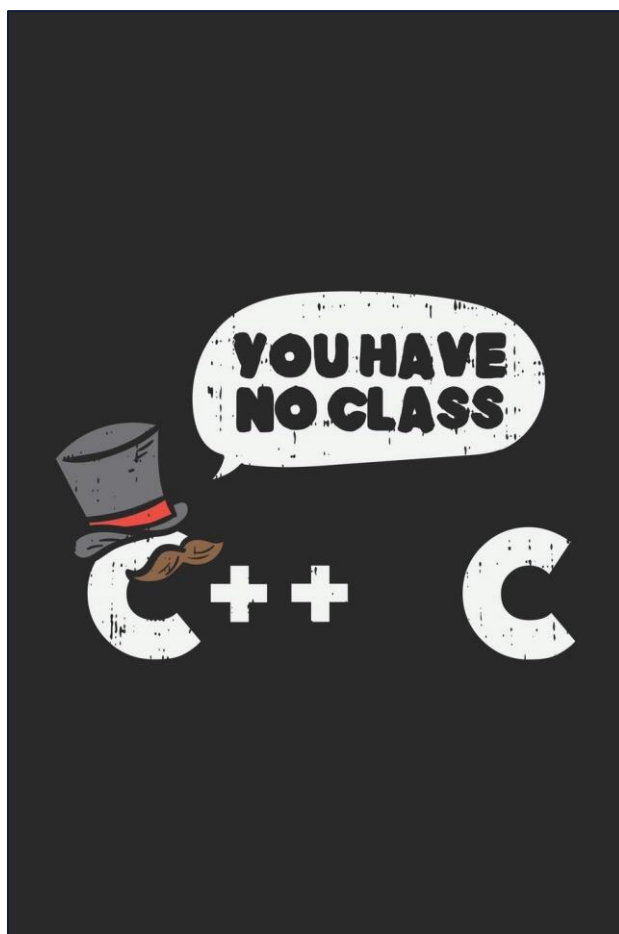
אופרטור שימושי נוסף הוא אופרטור החיבור '+', אשר מאפשר לנו לחבר מחרוזת עם תו, או מחרוזת עם מחרוזת אחרת. ניתן גם לחבר כמה מחרוזות ברצף, העיקר שלפחות אחד מבין שני גורמים שחיברנו הוא אובייקט מסוג `std::string`

```
std::string s1 = "Donald";
std::string s2 = "123";
std::string s3 = s1 + s2; // s3 = "Donald123"

std::string s4 = "Banel";
std::string s5 = "Static";
std::string s6 = s4 + " & " + s5; // s6 = "Banel & Static"
std::string s7 = s5 + " & Banel"; // s7 = "Static & Banel"
std::string s8 = "Banel & " + s5; // s8 = "Banel & Static"

std::string str1 = "Yofi " + "Tofi"; // compilation error!!!
```

המחלקה חושפת החוצה ממשק עשיר עם מתודות מאוד שימושיות, [עברו עליו](#), בהמשך התרגיל הממשק עשוי להקל עליכם/ן.



## שלב 3: רשת חברתית – מחלקת Device

שלב 6 מחלקת Social Network	שלב 5 מחלקות Profile ו- Page	שלב 4 מחלקת User	<b>שלב 3 מחלקת Device</b>	שלב 2 מחזזות	שלב 1 האופרטור &
-------------------------------	---------------------------------	---------------------	-------------------------------	-----------------	---------------------

בתרגיל נבנה רשת חברתית צנועה שבה משתמשים יוכלו להתחבר לפרופיל שלהם באמצעות מכשירים שונים, לכתוב פוסטים, ולהוסיף חברים. אנחנו נתחיל בליצור מחלקה המייצגת מכשיר (Device), ממנה נמשיך למחלקה שמייצגת משתמש/ת (User), אחר כך נמשיך למחלקת Profile ובסוף נבנה את המחלקה Social Network שתשתמש בכל המחלקות הקודמות שבנינו.

### מחלקת Device – רקע

המשתמשים של הרשת שלנו יוכלו להתחבר לרשת ממכשירים שונים, מכשיר יכול להיות מחשב נייד או לפטופ, טאבלט או טלפון. המחלקה הראשונה שנכתוב תהיה מחלקת Device אשר מייצגת מכשיר שמתחבר לרשת שלנו.

### מחלקת Device

לרשותכם/ן הקובץ `Device.h`, שם תוכלו למצוא ערכים קבועים שהוגדרו מראש כדי שיהיה נוח לבדוק את המחלקה.

- `enum DeviceType` – מגדיר 4 סוגי מכשירים – {מחשב נייד, לפטופ, טאבלט, טלפון}.
- הגדרות (`#define`) לסוגי מערכות ההפעלה שרצות על המכשירים.

אם תרצו תוכלו להוסיף סוגים נוספים, אבל אין למחוק את הסוגים הקיימים.

השלימו את הקובץ `Device.h`: רשמו את חתימות המתודות של המחלקה והצהירו על השדות שלה. שימו , בטבלה הבאה רשומות המתודות של המחלקה Device, תוכלו באמצעותה גם להסיק בעצמכם/ן מה השדות שהמחלקה צריכה להחזיק.

מכשיר - Device	
תיאור	מתודה/פעולה
מתחלת אובייקט של מכשיר חדש. <b>הערה:</b> הפונקציה תיקרא לפני השימוש באובייקט.	<code>init(unsigned int id, DeviceType type, std::string os)</code>
מחזירה את המספר המזהה של המכשיר.	<code>unsigned int getID()</code>
מחזירה את סוג המכשיר. <b>הערה:</b> הערך החוזר מהמתודה הוא מסוג <code>DeviceType</code> שהוגדר מראש בקובץ התרגיל.	<code>DeviceType getType()</code>

מכשיר - Device (המשך...)	
מתודה/פעולה	תיאור
<code>std::string getOS()</code>	מחזירה את שם מערכת ההפעלה של המכשיר. <b>הערה:</b> הערך החוזר מהמתודה הוא מסוג <code>std::string</code> שעליו למדנו בחלק הקודם.
<code>bool isActive()</code>	מחזירה האם המכשיר במצב "פועל" <b>הערה:</b> מכשיר חדש מתחיל במצב "פועל"
<code>void activate()</code>	משנה את מצב המכשיר ל-"פועל"
<code>void deactivate()</code>	משנה את מצב המכשיר ל-"לא פועל"

אחרי שסיימתם/ן את קובץ ההגדרות, **צרו קובץ חדש** בשם `Device.cpp` וממשו את המתודות שהוגדרו בקובץ `h`.

## הנחיות והערות חשובות:

- בהוראות לא רשומות החתימות המדויקות של המתודות של המחלקה, אבל האחריות שלכם/ן היא להוסיף `const` ולקבל פרמטרים כ-reference היכן שצריך.
- חשפו החוצה מהמחלקה כמה שפחות מידע: מה שאפשר הגדירו `private`, ומה שנרצה לחשוף החוצה הגדירו `public`.
- רשמו בקובץ ה-H כמה שפחות הגדרות וחתימות.  
לדוגמה, לרוב אין צורך לעשות `#include <iostream>` בקובץ ה-H, ניתן לכתוב זאת בקובץ המימוש (`cpp`).
- זכרו שכדי להשתמש במחלקת `std::string` יש לבצע `#include <string>`.
- מומלץ לבדוק את המחלקה לפני שממשיכים לסעיף הבא, תוכלו להיעזר בקובץ `test1Device.cpp` שניתן יחד עם קבצי התרגיל.
- ⚠️ חשוב! ⚠️  
הקפידו לתת שמות נכונים לכל דבר שאתם/ן יוצרים/ות, כלומר ע"פ מה שרשום בהוראות. הטסטים שבעזרתם ייבדק התרגיל משתמשים בשמות האלו והם לא יעברו אם השמות של הקבצים/מחלקות/מתודות שונים מההוראות.
- שימו ❤️, בקובץ `Tester.cpp` יש בדיקות לכל שלבי התרגיל, ואם לא מימשנו את כל התרגיל הקובץ לא יתקמפל. תוכלו להתגבר על הבעיה בשתי דרכים:
  - אל תשתמשו ב-`Tester.cpp` השתמשו בקובץ הבדיקות של חלק 1 – `test1Device.cpp`
  - שימו את כל שאר הקוד בהערה ותעבדו ישירות עם הקובץ `Tester.cpp`.



## שלב 4: רשת חברתית – מחלקת User

שלב 6 מחלקת Social Network	שלב 5 מחלקות Profile ו- Page	שלב 4 מחלקת User	שלב 3 מחלקת Device	שלב 2 מחזזות	שלב 1 האופרטור &
-------------------------------	---------------------------------	---------------------	-----------------------	-----------------	---------------------

### מחלקת User – רקע


השלב הבא ביצירת הרשת החברתית הוא לייצג משתמש של המערכת. כדי להירשם לרשת משתמש צריך ת.ז, שם משתמש וגיל. למשתמש יש רשימת מכשירים שאיתם הוא מתחבר לרשת.

### מחלקת User


בחלק הזה ניצור שני קבצים חדשים – `User.h` ו-`User.cpp`

בנוסף, בשלב הזה נשתמש ברשימה מקושרת שנכתבה ומומשה עבורנו בקבצים `DeviceList.h` ו-`DeviceList.cpp`.


כתבו את הגדרות המחלקה וממשו אותה ע"פ התיאור. שימו, ♥, בטבלה הבאה רשומות המתודות של המחלקה `User`, תוכלו באמצעותה גם להסיק בעצמכם/מה השדות שהמחלקה צריכה להחזיק.

<div>  <b>משתמש - User</b> </div>	
תיאור	מתודה/פעולה
מאתחלת אובייקט של משתמש חדש. <b>הערה:</b> הפונקציה תיקרא לפני השימוש באובייקט.	<code>void init(unsigned int id, std::string username, unsigned int age)</code>
מנקה את האובייקט ומוחקת זיכרון שהוקצה לאובייקט. <b>הערה:</b> אם במהלך יצירת האובייקט איתחלנו אובייקטים אחרים, אז בפונקציה זו אנו ננקה גם אותם. אולם אין צורך להשתמש ב- <code>delete</code> אלא אם כן הייתה הקצאת זיכרון דינמית באמצעות המילה <code>new</code> .	<code>void clear()</code>
מחזירה את תעודת הזהות של המשתמש.	<code>unsigned int getID()</code>
מחזירה את שם המשתמש.	<code>std::string getUsername()</code>
מחזירה את גיל המשתמש.	<code>unsigned int getAge()</code>



<div>  <b>משתמש - User (המשך...)</b> </div>	
תיאור	מתודה/פעולה
<p>מחזירה הפניה לרשימת המכשירים של המשתמש.</p> <p><b>הערה:</b> הסוג החוזר מהפונקציה הוא reference שעליו למדנו בתחילת התרגיל. חשבו למה אנחנו רוצים להחזיר reference ולא עותק...</p>	<pre>DevicesList&amp; getDevices();</pre>
<p>מוסיפה מכשיר לרשימת המכשירים של המשתמש.</p> <p><b>הערה:</b> הפונקציה צריכה לעשות שימוש במחלקה <code>DeviceList</code> שניתנה עם קבצי התרגיל.</p>	<pre>void addDevice(Device newDevice);</pre>
<p>מחזירה <code>true</code> אם כל המכשירים שבבעלות המשתמש נמצאים במצב "פועל".</p> <p><b>הערה:</b> הפונקציה צריכה לעשות שימוש במחלקה <code>DeviceList</code> שניתנה עם קבצי התרגיל.</p>	<pre>bool checkIfDevicesAreOn()</pre>

## הנחיות והערות חשובות:

- בהוראות לא רשומות החתימות המדויקות של המתודות של המחלקה, אבל האחריות שלכם/ היא להוסיף `const` ולקבל פרמטרים כ-reference היכן שצריך.
- זכרו לבצע `#include "DeviceList.h"` כדי שתוכלו להשתמש ברשימה שסופקה לכם/ן.
- שימו , מומלץ לבדוק את המחלקה לפני שממשיכים לסעיף הבא, תוכלו להיעזר בקובץ `test2User.cpp` שניתן יחד עם קבצי התרגיל.



## שלב 5: רשת חברתית – מחלקות Profile ו-Page

שלב 6 מחלקת Social Network	שלב 5 מחלקות Profile ו-Page	שלב 4 מחלקת User	שלב 3 מחלקת Device	שלב 2 מחזרות	שלב 1 האופרטור &
-------------------------------	--------------------------------	---------------------	-----------------------	-----------------	---------------------

### מחלקות Profile ו-Page – רקע

עבשיו נוכל להתקדם שלב נוסף, וליצור מחלקה המייצגת פרופיל ברשת החברתית שלנו. לכל פרופיל יש בעלים – המשתמש (User) שהפרופיל שייך לו/ה. בתוך הפרופיל יש עמוד (Page) שבו אפשר לפרסם סטטוס ופוסטים, וגם רשימת חברים (משתמשים אחרים).

### מחלקת Page

נתחיל במחלקה Page, המייצגת עמוד בפרופיל של משתמש מסוים. העמוד מכיל את הסטטוס של המשתמש ואת הפוסטים שלו. בחלק הזה ניצור 2 קבצים חדשים - `Page.h`, `Page.cpp`

בתוך עמוד נחזיק שני שדות מסוג מחזרות:

- עבור הסטטוס שיהיה רשום בראש העמוד.
  - עבור הפוסטים שכתובים בעמוד.
- פוסט יכול להיות משפט או רצף של מילים שהמשתמש מעוניין לשתף. כל הפוסטים מאוחסנים יחד במחזרות אחת, בכל פעם שמתווסף פוסט לעמוד, נוספת שורה חדשה למחזרות הפוסטים.

ממשו את המחלקה Page ע"פ התיאור הבא, הקפידו להגדיר את שני השדות שתוארו למעלה:

עמוד - Page	
מחזרות/פעולה	תיאור
<code>init()</code>	מאתחלת אובייקט של עמוד חדש עם מחזרות סטטוס ריקה ומחזרות פוסטים ריקה. <b>הערה:</b> הפונקציה תיקרא לפני השימוש באובייקט.
<code>std::string getPosts()</code>	מחזירה מחזרות עם הפוסטים שבעמוד, כל פוסט בשורה חדשה: לדוגמה:  <code>Hello world!</code> <code>Good morning :)</code> <code>Magshimim forever ***</code>
<code>std::string getStatus()</code>	מחזירה את הסטטוס. <b>הערה:</b> הסטטוס זה מחזרות שתמיד מופיעה בראש העמוד.

<code>void clearPage()</code>	מנקה את הפוסטים מעמוד הפרופיל. <b>הערה:</b> המתודה מאפסת את מחזורת הפוסטים, אבל לא משנה את הסטטוס של המשתמש.
<code>void setStatus(std::string status)</code>	משנה את הסטטוס.
<code>void addLineToPosts(std::string new_line)</code>	מוסיפה שורה חדשה למחזורת הפוסטים.

## מחלקת Profile

כעת נעבור למחלקה Profile אשר מחזיקה את המידע על המשתמש ברשת החברתית.


בחלק הזה ניצור 2 קבצים חדשים – `Profile.h`, `Profile.cpp` בנוסף, בשלב הזה נשתמש ברשימה מקושרת שנכתבה ומומשה עבורנו בקבצים `UserList.h` ו-`UserList.cpp`.

בתוך פרופיל נחזיק שלושה שדות:

- המשתמש/ת (`User`) שהפרופיל בבעלותו/ה.
- עמוד (`Page`).
- רשימת חברים (`UserList`).

הגדירו את המחלקה וממשו אותה ע"פ התיאור הבא:

פרופיל – Profile	
<code>void init(User owner);</code>	מאתחלת אובייקט של פרופיל חדש. <b>הערה:</b> הפונקציה תיקרא לפני השימוש באובייקט.
<code>void clear()</code>	מנקה את האובייקט ומוחקת זיכרון שהוקצה לאובייקט. <b>הערה:</b> אם במהלך יצירת האובייקט אתחלנו אובייקטים אחרים, אז בפונקציה זו אנו ננקה גם אותם. אולם אין צורך להשתמש ב- <code>delete</code> אלא אם כן הייתה הקצאת זיכרון דינמית באמצעות המילה <code>new</code> .
<code>User getOwner()</code>	מחזירה העתק של אובייקט המשתמש שהפרופיל בבעלותו.
<code>void setStatus(std::string new_status)</code>	משנה את הסטטוס שבעמוד של בעל/ת הפרופיל לסטטוס החדש. <b>הערה:</b> המתודה משתמשת במתודה <code>Page::setStatus()</code> .
<code>void addPostToProfilePage(std::string post)</code>	מוסיפה פוסט לעמוד של בעל/ת הפרופיל. <b>הערה:</b> המתודה משתמשת במתודה <code>Page::addLineToPage()</code> .
<code>void addFriend(User friend_to_add)</code>	מוסיפה חבר/ה לרשימת החברים של הפרופיל. <b>הערה:</b> הפונקציה צריכה לעשות שימוש במחלקה <code>UserList</code> שניתנה עם קבצי התרגיל.

 פרופיל – Profile (המשך...)	
<pre>std::string getPage();</pre>	<p>מחזירה מחרוזת עם הסטטוס והפוסטים שבעמוד של בעל/ת הפרופיל. הפורמט של המחרוזת הוא כזה:</p> <pre>Status: &lt;User's status&gt; ***** ***** &lt;post1&gt; &lt;post2&gt; &lt;post3&gt; ...</pre> <p>דוגמה:</p> <pre>Status: Status: Which witch watched which watch??? ***** ***** Hello world! Good morning :) Magshimim forever ***</pre>
<pre>std::string getFriends();</pre>	<p>מחזירה מחרוזת עם שמות החברים של בעל/ת הפרופיל מופרדים בפסיק. הפורמט של המחרוזת הוא כזה:</p> <pre>&lt;friend1&gt;,&lt;friend2&gt;,&lt;friend3&gt;,...,&lt;friendn&gt;</pre> <p>דוגמה:</p> <pre>Avi,Nitzan,Rinat,Shlomi</pre> <p><b>הערה:</b> הפונקציה צריכה לעשות שימוש במחלקה <code>UserList</code> שניתנה עם קבצי התרגיל.</p>
<pre>std::string getFriendsWithSameNameLength()</pre>	<p>מחזירה מחרוזת עם רשימת החברים ששם המשתמש שלהם הוא באותו האורך כמו שם המשתמש של הבעלים של הפרופיל. לדוגמה:</p> <p>נניח שהשם של הבעלים של הפרופיל הוא "Shlomo" ושרשימת החברים היא: "Avner,Nitzan,Motti,Maya,Lilach"</p> <p>המתודה תחזיר את המחרוזת הבאה: "Nitzan,Lilach"</p>

## הנחיות והערות חשובות:

- בהוראות לא רשומות החתימות המדויקות של המתודות של המחלקה, אבל האחריות שלכם/ן היא להוסיף `const` ולקבל פרמטרים כ-reference היכן שצריך.
- זכרו לבצע `#include "UserList.h"` כדי שתוכלו להשתמש ברשימה שסופקה לכם/ן.

- המחזרות שחוזרת מהמתודות `getFriends()` , `getFriendsWithSameNameLength()` לא מכילה פסיק ';' , בסוף המחזרות, חשבו איך לגרום לתו לא להופיע.
  - שימו ♥ , מומלץ לבדוק את המחלקה לפני שממשיכים לסעיף הבא, תוכלו להיעזר בקובץ `test3Profile.cpp` שניתן יחד עם קבצי התרגיל.
  - !!! הערה בקשר לרשימות המקושרות הניתנות בתרגיל !!!
- אם תעברו על הקוד שבקבצים שסופקו יחד עם הוראות התרגיל תראו שיש כפל קוד. כל מחלקה שמייצגת רשימה בתרגיל זהה כמעט לחלוטין למחלקות האחרות, והדבר היחיד ששונה זה סוג האיברים שכל רשימה מחזיקה.
- חשוב שבבין שקוד שחוזר על עצמו בצורה כזו הוא לא קוד יעיל, ולאורך הקורס ננסה לא לחזור על עצמנו פעמיים ולא להשתמש ב-copy & paste יותר מידי... אבל כרגע בשלב הזה של הקורס אין לנו כלים לפתור את זה. בשבוע 9 נלמד על מנגנון שמאפשר להגדיר מחלקות כלליות, שלא חשוב איזה סוג משתנים הן מחזיקות, והקוד ייכתב עבורנו אוטומטית.





החלק הבא אינו חובה, אבל זה תרגול מצוין בעבודה עם מחרוזות.

הועסקתם/ן ע"י ארגון בטחוני גדול, שביקש מכם להכניס שתי פונקציות למחלקה [Profile](#). בתור מרגלים ומרגלות, תצטרכו לממש שתי פונקציות משעשעות במיוחד, שמשנות את הסטטוס בפרופיל וגורמות לדברים להישמע אחרת ממה שהתכוונו...

<pre>void changeAllWordsInStatus(std::string word)</pre>	<p>משנה את כל המילים בסטטוס למילה אחת מסוימת.</p> <p>דוגמה: נניח ויש לנו פרופיל שבעמוד שלו יש סטטוס:</p> <p>Have you ever wondered why you can't taste your tongue? That's crazy!!!</p> <p>אם נקרא למתודה עם המילה "Magshimim" הסטטוס ישתנה ל:</p> <p>Magshimim Magshimim Magshimim Magshimim Magshimim Magshimim Magshimim Magshimim Magshimim Magshimim Magshimim Magshimim</p> <p>שימו , אם במחרוזת המקורית היו רווחים, טאבים או ירידות שורה הם צריכים להופיע גם בסטטוס החדש.</p>
<pre>void changeWordInStatus(std::string word_to_replace,                         std::string new_word)</pre>	<p>מחליפה את כל המופעים של מילה מסוימת במחרוזת, למילה אחרת.</p> <p>דוגמה: נניח ויש פרופיל עם עמוד שבו הסטטוס הוא:</p> <p>No matter how smart you are, you can never convince someone stupid that they are stupid.</p> <p>אם נקרא למתודה כדי להחליף את המילה "you" עם המילה "MMMMMMMMMMMM" הסטטוס ישתנה ל:</p> <p>No matter how smart MMMMMMMMMMMM are, MMMMMMMMMMMM can never convince someone stupid that they are stupid.</p>

את הפונקציות ניתן לבדוק ע"י הרצת הטסט test3Bonus שבקובץ [test3Profile.cpp](#)

## שלב 6: רשת חברתית – מחלקת Social Network

שלב 6 מחלקת Social Network	שלב 5 מחלקות Profile ו- Page	שלב 4 מחלקת User	שלב 3 מחלקת Device	שלב 2 מחזרות	שלב 1 האופרטור &
-------------------------------	------------------------------------	---------------------	-----------------------	-----------------	---------------------

### מחלקת Social Network – רקע


וואו זה היה ארוך... אבל עכשיו סוף סוף יש לנו את כל החלקים ואנחנו יכולים להגדיר את המחלקה האחרונה והחשובה ביותר – `SocialNetwork` אשר מייצגת רשת חברתית.


### מחלקת SocialNetwork

בחלק הזה ניצור 2 קבצים חדשים – `SocialNetwork.h` ו-`SocialNetwork.cpp` בנוסף, בשלב הזה נשתמש ברשימה מקושרת שנכתבה ומומשה עבורנו בקבצים `ProfileList.h` ו-`ProfileList.cpp`.

המחלקה מייצגת רשת חברתית, שמחזיקה את שם הרשת, רשימה של פרופילים וגיל מינימלי שאסור שהמשתמשים שלה יהיו צעירים ממנו.

הגדירו את המחלקה וממשו אותה ע"פ התיאור הבא:

<div>  Social Network – רשת חברתית </div>	
מתודה/פעולה	תיאור
<code>void init(std::string networkName, int minAge)</code>	מאתחלת אובייקט של רשת חברתית. <b>הערה:</b> הפונקציה תיקרא לפני השימוש באובייקט.
<code>void clear()</code>	מנקה את האובייקט ומוחקת זיכרון שהוקצה לאובייקט. <b>הערה:</b> אם במהלך יצירת האובייקט איתחלנו אובייקטים אחרים, אז בפונקציה זו אנו ננקה גם אותם. אולם אין צורך להשתמש ב- <code>delete</code> אלא אם כן הייתה הקצאת זיכרון דינמית באמצעות המילה <code>new</code> .
<code>std::string getNetworkName();</code>	מחזירה מחזרות עם שם הרשת
<code>int getMinAge();</code>	מחזירה את הגיל המינימלי שממנו אפשר להצטרף לרשת.
<code>bool addProfile(Profile profile_to_add)</code>	מוסיפה פרופיל חדש לרשימת הפרופילים של הרשת החברתית. <b>הערה:</b> אם הפרופיל בבעלות של משתמש צעיר מהגיל המינימלי המותר ברשת, הפונקציה צריכה להחזיר <code>false</code> ולא להוסיף את הפרופיל לרשת.  <b>הערה נוספת:</b> הפונקציה צריכה לעשות שימוש במחלקה <code>ProfileList</code> שניתנה עם קבצי התרגיל.

רשת חברתית – Social Network (המשך...)	
מתודה/פעולה	תיאור
<code>std::string getWindowsDevices()</code>	<p>מחזירה מחרוזת עם רשימה של מכשירים שמריצים מערכת הפעלה Windows ומחוברים לרשת החברתית. עבור כל מכשיר יתווסף למחרוזת המספר המזהה שלו ומערכת ההפעלה שלו, אח"כ פסיק ואז רווח.</p> <p>דוגמה:</p> <p><code>[2123, Windows11], [1121, Windows10], [5361, Windows7], [7711, Windows11], [9444, Windows11]</code></p> <p>שימו , עבור כל מכשיר נכניס למחרוזת סוגריים מרובעים שבתוכן המספר המזהה של המכשיר, רווח ופסיק, ואת מערכת ההפעלה שלו.</p>

## הנחיות והערות חשובות:

- בהוראות לא רשומות החתימות המדויקות, אבל האחריות שלכם/ן היא להוסיף `const` ולקבל פרמטרים כ- reference היכן שצריך.
- המחרוזת שחוזרת מהמתודה `getWindowsDevices()` לא מכילה פסיק ';' ורווח בסוף, חשבו איך לגרום לתווים לא להופיע.
- זכרו לבצע `#include "ProfileList.h"` כדי שתוכלו להשתמש ברשימה שסופקה לכם/ן.
- מומלץ לבדוק את המחלקה לפני שממשיכים לסעיף הבא, תוכלו להיעזר בקובץ `Tester.cpp` שביתן יחד עם קבצי התרגיל. בתוך ה-Tester ניתן להריץ את הבדיקה `.test4SocialNetwork()`.

שימו , עכשיו כבר ניתן להריץ את ה-Tester במלואו, אם כל הטסטים עברו תקבלו את המסך הבא.



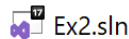
במידה והייתה בעיה עם אחד הטסטים, ה-Tester ידפיס הודעת שגיאה ויכווין למתודות שעליהן בדאי להסתכל.



## נספחים

### הגשה ב-GIT

- את הפרויקט יש לנהל ב-Git, לפתוח repository חדש בתוך קבוצת ה-gitlab שלנו ושל המדריך/ה, ולהגיש לינק לפרויקט ב-NEO (אפשר לעשות comment עם הלינק או להגיש מסמך txt עם הלינק בפנים).
- יש להעלות ל-repository את 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](#)), 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**[בסרטוני עזר בנושא GIT](#)**.
- כסיימתם/ן, בדקו שניתן להריץ 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:

- את הפרויקט יש לפתוח בקבוצת ה-gitlab שאליה משותף/ת המדריך/ה כ-Maintainer.
- 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 בסרטוני ההדרכה בנושא GIT. ניתן לגשת לסרטונים בלשונית ה-resources שבכיתה ה-NEO.
- בסיום העבודה יש להגיש לכיתה ה-NEO קישור ל-repository.

### כללי

- יש לבדוק שכל **המטלות מתקמפלות ורצות ב-VS2022**. מטלה שלא תעבור קומפילציה אצל הבודק לא תיבדק **והניקוד שלה יהיה 0** 😞
- 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.