מכללת הדסה, החוג למדעי המחשב

מבוא לתכנות מונחה עצמים והנדסת תוכנה

סמסטר ב', תשע"ט

תרגיל 1

תאריך אחרון להגשה:

יום ב', 25/03/2019, בשעה 23:59

מטרות התרגיל:

בתרגיל זה נמשיך לעסוק בתכנון ממשק ונושאים אחרים שכבר עסקנו בהם, אולם הפעם נתמקד במיוחד במחלקות העוסקות בהקצאות זיכרון דינמיות, שימוש נכון בבנאי העתקה (Copy Constructor), אופרטור השמה (Assignment Operator), העמסת אופרטורים (Operator Overloading), ובהורס (Destructor).

תיאור כללי:

בתרגיל זה נגדיר מחלקה Image המייצגת תמונה בגוונים של שחור/לבן/אפור בלבד. התמונה תהיה בגובה H וברוחב W כלשהם, כאשר H ו־W מתקבלים בבנאי.

כמו כן נגדיר מחלקה Pixel המייצגת תא אחד של התמונה. כלומר לתמונה בגודל W*H יהיו W*H אובייקטים של Pixel. ל-Pixel נגדיר 3 צבעים אפשריים, צבע שחור, אפור ולבן. אין דבר כזה Pixel ללא צבע.

צבע שחור יוגדר על ידי התו 219. ניתן להגיע אליו כך:

```
const unsigned char BLACK = (unsigned char)219
```

צבע אפור יוגדר על ידי התו 176 ניתן להגיע אליו כך:

```
const unsigned char GRAY = (unsigned char)176
```

צבע לבן יוגדר על ידי התו רווח ' ' (תו מספר 32).

מכיוון שאחת ממטרות התרגיל היא כדי שתתרגלו שימוש נכון בהקצאות זיכרון דינמיות, אין להשתמש בתרגיל זה במחלקות מוכנות כגון std::list, std::vector או std::string, שתעקופנה את הצורך שלכם ליצור את הפונקציונליות הזו בעצמכם (יש חריגה אחת לגבי std::string כפי שתובא בהמשך באופרטור ההוצאה). שימו לב שבהורס (destructor) אתם צריכים לשחרר את כל ההקצאות שעשיתם.

נאסר עליכם להשתמש ב-malloc/realloc/free, אלא עליכם להשתמש ב־new/new[]/delete/delete[].

מחלקת מבנה נתונים:

נגדיר מחלקה בשם ImageDataStructure שתשמש אותנו לטיפול בהקצאות הזיכרון ושיחרורו כמתבקש. שימו לב כי אתם חייבים להשתמש בה (ולא להקצות את הזיכרון של התמונה ישר בתוך מחלקת Image), והיא תקל עליכם בבואכם לממש את מחלקת Image שתובא בהמשך.

הערות:

אתם נדרשים לחשוב גם על שיקולים של עיצוב מונחה עצמים (OOD), לא פחות ואולי אף יותר מאשר על שיקולים של יעילות. למשל, ברוב האופרטורים שאתם נדרשים להגדיר, ניתן וראוי לעשות שימוש באופרטורים אחרים, לרוב בסיסיים יותר. אתם רשאים להוסיף מחלקות כפי הצורך.

עוד הערה חשובה: אמנם רוב הגדרת התרגיל עוסקת בהגדרת האופרטורים שעליכם לממש, אולם בפועל, עבודת התכנות הנדרשת עבורם אמורה להסתכם בשורות קוד בודדות, לעיתים שורה אחת, עבור כל אופרטור. רוב העבודה בתרגיל זה מתרכזת סביב הגדרת מבנה הנתונים (עם קביעת ממשק המחלקה או המחלקות למימושו) וניהול נכון של הזיכרון.

הערה אחרונה: אין בתרגיל שום דרישה לגבי פונקציית main. זו לא טעות. אתם יכולים לכתוב לעצמכם פונקציית main כדי להריץ את התוכנית ולבדוק את המימוש שלכם (מומלץ), אבל אין דרישה להגיש אותה. המחלקה שלכם אמורה לעבוד עם כל פונקציית main שתשתמש בממשק שמוגדר כאן בתרגיל.

פירוט הדרישות:

בנאים שאתם נדרשים לממש:

Image()

– בנאי שיאתחל את התמונה הריקה, שאין בה תאים כלל.

Image(int height, int width)

– בנאי שיאתחל את התמונה לגובה ולרוחב הנתונים. בכל התאים, כלומר הפיקסלים, יהיה צבע לבן.

בנאי העתקה:

Image(const Image& other)

– בנאי שיעתיק את התמונה המתקבלת לתוך האובייקט החדש. חשוב לציין שחייבת להיות כאן העתקה עמוקה, כך ששינוי באובייקט אחד לא ישפיע כלל על האובייקט השני.

על מחלקת Pixel ועל מחלקת ImageDataStructure אין דרישה לבנאים מסוימים. אתם יכול להגדיר ולממש את הבנאים כפי הצורך.

אופרטורים שאתם נדרשים לממש עבור המחלקה Pixel:

לכל ההגדרות של האופרטורים נניח 2 פיקסלים P1 ו-P2.

אופרטורים **בינאריים** שאתם נדרשים לממש:

- P1==P1, P1!=P2 – השוואת פיקסלים. יחזיר true אם ורק אם הפיקסלים שווים בצבעם. (-זה במקרה של שוויון. במקרה של אי־שוויון יחזיר כמובן false).
- <<P1 – אופרטור ההכנסה (Insertion Operator) – מכניס (מדפיס) אל ה־stream את ה־Pixel. על ידי כך אם נריץ את הפקודה std::cout<<P1 הפיקסל P1 יודפס למסך, שכן std::cout הוא אובייקט מסוג std::ostream.

  תזכורת: חתימת הפונקציה למימוש האופרטור הזה היא:

```
std::ostream& operator<<(std::ostream& os, const Pixel& image);
```

- P1|P2 – איחוד של הפיקסל P1 והפיקסל P2. התוצאה תהיה כך: אם הפיקסלים שווים – האיחוד שלהם יתן פיקסל עם אותו ערך. אם הפיקסלים שונים, האיחוד שלהם יתן את פיקסל עם הערך של התא **הכהה** יותר. **נא לשים לב:** הפיקסלים P1 ו־P2 אינם משתנים כלל. האופרטור מחזיר את התוצאה, אבל לא משנה כלל את הפיקסלים עצמם.
- P1|=P2 – איחוד והשמה של פיקסלים. פעולה זה שקולה לפעולה: P1=P1|P2.
- P1&P2 – חיתוך של הפיקסל P1 והפיקסל P2. התוצאה תהיה כך: אם הפיקסלים שווים – החיתוך שלהם יתן פיקסל עם אותו ערך. אם הפיקסלים שונים, החיתוך שלהם יתן את פיקסל עם הערך של התא **הבהיר** יותר. **נא לשים לב:** הפיקסלים P1 ו־P2 אינם משתנים כלל. האופרטור מחזיר את התוצאה, אבל לא משנה כלל את הפיקסלים עצמם.
- P1&=P2 – חיתוך והשמה של פיקסלים. פעולה זה שקולה לפעולה: P1=P1&P2.

אופרטורים שאתם נדרשים לממש עבור המחלקה Image:

לכל ההגדרות של האופרטורים נניח שקיימות 2 תמונות A ו־B. תמונה A בגובה Ha וברוחב Wa, ותמונה B בגובה Hb וברוחב Wb.

אופרטורים **בינאריים** שאתם נדרשים לממש:

- A==B, A!=B – השוואת תמונות. יחזיר true אם ורק אם גם הגובה, גם הרוחב, וגם כל הפיקסלים של התמונות זהה (-זה במקרה של שוויון. ובמקרה של אי־שוויון יחזיר כמובן false).

- A=B – השמה של תמונה. התמונה B תועתק במלואה לתמונה A. שימו לב שאתם מנהלים נכון את ההקצאות ו/או השחרור של הזיכרון של תמונה A כך שלא יהיה לכם דליפת זיכרון. כפי שהזכרנו בבנאי העתקה: חשוב לציין שחייבת להיות כאן העתקה עמוקה, כך ששינוי באובייקט אחד לא ישפיע כלל על האובייקט השני.

  נא לשים לב: אופרטור ההשמה דומה במימושו לבנאי העתקה. תוודאו שאין לכם כאן קוד כפול.

- A+B – חיבור של שתי תמונות. התוצאה תהיה תמונה גדולה בגובה max(Ha,Hb), וברוחב Wa+Wb, ולמעשה תשרשר את שתי התמונות. אם התמונות אינן באותו גודל, התמונה הנמוכה יותר תהיה צמודה מעלה, ובמקומות "הריקים" נמלא את הצבע הלבן.

  נא לשים לב: התמונות A ו־B אינן משתנות כלל. האופרטור מחזיר את התוצאה, אבל לא משנה כלל את התמונות עצמן.

  נא לשים לב: פעולת החיבור שהגדרנו אינה קומוטטיבית. כלומר לא בהכרח מתקיים ש: A+B==B+A. הדבר הזה בסדר, שכן גם במחלקה std::string ההתנהגות דומה. למשל אם str1!=str2 אזי בדרך כלל str1+str2!=str2+str1.

דוגמא לחיבור של התמונות. חיבור של התמונה הזו (רק לשם הדוגמה הגדרנו כאן לבן '-', שחור '#' ואפור '*'):

```
----------
--**--**--
---****---
----**----
----**----
```

יחד עם התמונה הזו:

```
-**----
--**---
---**--
----**-
```

יתן את התוצאה הזו:

```
----------**----
```

```
__**__**____**___
___****______**__
____**________**_
____**__________
```

- A+=B – חיבור והשמה של תמונה. פעולה זה שקולה לפעולה: A=A+B.
- <<A – אופרטור ההכנסה (Insertion Operator) – מכניס (מדפיס) אל ה־stream את Image. על ידי כך אם נריץ את הפקודה std::cout<<A התמונה A תודפס למסך, שכן std::cout הוא אובייקט מסוג std::ostream.

```
std::ostream& operator<<(std::ostream& os, const Image& image);
```

- >>A – אופרטור ההוצאה (Extraction Operator) – מוציא (מקבל קלט) מה־stream אל ה־Image. בתחילת הסטרים צריכים להופיע שני מספרים: הראשון יהיה הגובה של התמונה, והשני יהיה הרוחב של התמונה. ואז קוראים שורה אחר שורה לפי הממדים שהוגדרו. על ידי כך אם נריץ את הפקודה std::cin>>A נוכל לעדכן את התמונה A, שכן std::cin הוא אובייקט מסוג std::istream. **שימו לב:** שהפעולה הזו גורמת **לדריסת התמונה הקיימת** (אם A אינה ריקה), כלומר לשינוי הגודל שלה לגודל שהתקבל ולשינוי התוכן שלה לתוכן שהתקבל מה־stream. זה אומר, בין השאר, שאולי תמצאו צורך בהגדרת האופרטור הזה כ־friend במחלקת Image.

הערה חשובה: באופרטור ההוצאה, זהו המקום היחידי בתרגיל שאתם רשאים להשתמש ב-std::string, שיכול לעזור לכם לקבל את הקלט מה-stream.

תזכורת: חתימת הפונקציה למימוש האופרטור הזה היא:

```
std::istream& operator>>(std::istream& os, Image& image);
```

- A|B – איחוד של התמונה A והתמונה B. התוצאה תהיה תמונה בגודל של מקסימום הרוחב ומקסימום הגובה של תמונות A ו־B. כל פיקסל ופיקסל של התוצאה יהיה הפיקסל המתאים של התמונה A "מאוחד" יחד עם הפיקסל המתאים של התמונה B. איחוד של 2 פיקסלים מוגדר לעיל באופרטורים של פיקסל. באזורים שיש רק תמונה אחת, ואין חפיפה בין התמונות (כלומר שאין לפיקסל של תמונה A מול מה לעשות איחוד או להפך) ניקח לתוצאה את הפיקסל של התמונה האחת כמו שהוא. באזורים שאין שם תמונה כלל, **נכניס פיקסלים לבנים**.

נא לשים לב: התמונות A ו־B אינן משתנות כלל. האופרטור מחזיר את התוצאה, אבל לא משנה כלל את התמונות עצמן.

נא לשים לב: פעולת האיחוד שהגדרנו קומוטטיבית. כלומר מתקיים ש: A|B==B|A.

ניתן להבין את פעולת האיחוד על ידי הדוגמא הבאה (שוב, רק לשם הדוגמה הגדרנו כאן לבן '-', שחור '#' ואפור '*'). איחוד של התמונה הזו (שיש בה 6 שורות ו-10 עמודות):

```
*-*-*-*-*-
----*-----
---***----
--*#*#*---
-*******--
----------
```

יחד עם התמונה הזו (שיש בה 7 שורות ו-9 עמודות):

```
---------
---------
-***#***-
--*****--
---*#*---
----*----
*-*-*-*-*
```

יתן את התוצאה הזו (תמונה עם 7 שורות ו-10 עמודות):

```
*-*-*-*-*-
----*-----
-***#***--
--*#*#*---
-***#***--
----*-----
-*-*-*-*-*
```

- A|=B – איחוד והשמה של התמונה. פעולה זה שקולה לפעולה: A=A|B.

- A&B – חיתוך של התמונה A והתמונה B. התוצאה תהיה תמונה בגודל של מינימום הרוחב ומינימום הגובה של תמונות A ו־B. כל פיקסל ופיקסל של התוצאה יהיה הפיקסל המתאים של התמונה A "חתוך" יחד עם הפיקסל המתאים של התמונה B. חיתוך של 2 פיקסלים מוגדר לעיל באופרטורים של פיקסל.

  נא לשים לב: התמונות A ו־B אינן משתנות כלל. האופרטור מחזיר את התוצאה, אבל לא משנה כלל את התמונות עצמן.

  נא לשים לב: פעולת החיתוך שהגדרנו קומוטטיבית. כלומר מתקיים ש: A&B==B&A.
- A&=B – חיתוך והשמה של התמונה. פעולה זה שקולה לפעולה: A=A&B.

ניתן להבין את פעולת החיתוך על ידי הדוגמא הבאה (שוב, רק לשם הדוגמה הגדרנו כאן לבן '-', שחור '#' ואפור '*'). החיתוך של התמונה הזו:

```
*-*-*-*-*
----*----
---***---
--*#*#*--
-*******-
---------
```

יחד עם התמונה הזו:

```
---------
---------
-***#***-
--*****--
---*#*---
----*----
*-*-*-*-*
```

יתן את התוצאה הזו:

```
---------
---------
---***---
--*****--
```

```
---***---
--------
```

- A*n – כפל בסקלר מימין, כאשר n מספר שלם מסוג unsigned int. נגדיר בפשטות כי A*n למעשה מחזיר תמונה השווה ל: A+…+A+A+A בדיוק n פעמים. במקרה ו־n שווה אפס, תחזירו תמונה ריקה.
- n*A – כפל בסקלר משמאל, מתנהג בדיוק כמו כפל בסקלר מימין.

נא לשים לב: בכפל בסקלר מימין ומשמאל, התמונה A אינה משתנה כלל. האופרטור מחזיר את התוצאה, אבל לא משנה כלל את התמונה עצמה.

- A*=n – כפל בסקלר והשמה של התמונה. פעולה זה שקולה לפעולה: A=A*n.

אופרטורים **אונריים** שאתם נדרשים לממש:

- A++ – הגדלת התמונה. התוצאה תהיה שתמונה A תישאר באותו גובה (כלומר בגובה Ha), אך היא תכפיל את רוחבה (כלומר ברוחב Wa*2). דבר זה יתבצע על ידי זה שכל שורה תגדל לפי 2 מהרוחב המקורי שלה, כאשר כל תא, כלומר פיקסל, בשורה "יוכפל".

  נא לשים לב: התמונה A בעצמה **כן** משתנית. האופרטור **גם** משנה את התמונה A עצמה **וגם** מחזיר הפניה (reference) של התמונה A עצמה כפי שהיא **לאחר** השינוי.

הדבר יובן בקלות על ידי ההמחשה הבאה. התמונה הזו:

```
----------
--**--**--
---****---
----**----
----**----
```

אחרי פעולת האופרטור, תהפוך לתמונה הזו:

```
--------------------
----****----****----
------********------
--------****--------
--------****--------
```

- A-- – הקטנת התמונה. התוצאה תהיה תמונה בגובה Ha וברוחב Wa/2, כלומר כל שורה תסתיים ברוחב של חלקי 2 מהרוחב המקורי שלה, וההקטנה תוגדר על ידי "דגימת כל פיקסל שני". הדבר יתבצע על ידי סריקת כל שורה משמאל, כאשר כל זוג תאים בשורה יקובצו לתא אחד בודד, על ידי בחירת התא השני בזוג, והתעלמות מהתא הראשון בזוג. על פי הגדרה זו, במקרה של מספר אי זוגי של תאים בשורה, התא האחרון בשורה יעלם. אפשר לחשוב על האופרטור הזה כך: שאנחנו "חותכים" מהתמונה את כל העמודות האי-זוגיות.

נא לשים לב: התמונה A בעצמה **כן** משתנית. האופרטור **גם** משנה את התמונה A עצמה **וגם** מחזיר הפניה (reference) של התמונה A עצמה כפי שהיא **לאחר** השינוי.

למשל התמונה הזו:

```
----------
--**--**--
---****---
----**----
----**----
```

אחרי פעולת האופרטור, תהפוך לתמונה הזו:

```
-----
-*-*-
-**--
--*--
--*--
```

והתמונה הזו (שמאוד דומה לתמונה הראשונה, רק שקיצרנו את הרוחב שלה ב-1):

```
---------
-**--**--
--****---
---**----
---**----
```

אחרי פעולת האופרטור, תהפוך לתמונה הזו:

```
----
*-*-
-**-
-*--
-*--
```

הערה: שימו לב שלאור ההגדרות הנ"ל מתקיים:

(--(++A))==A

- ++A – האופרטור הזה ישנה את התמונה A בדיוק כמו האופרטור A++. רק ההבדל הוא שהאופרטור הזה מחזיר **העתק** של התמונה A כפי שהייתה **לפני** השינוי. (בניגוד לאופרטור A++ שמחזיר את התמונה A **בעצמה**, כפי שהיא **לאחר** השינוי).
- --A – האופרטור הזה ישנה את התמונה A בדיוק כמו האופרטור A--. רק ההבדל הוא שהאופרטור הזה מחזיר **העתק** של התמונה A כפי שהייתה **לפני** השינוי. (בניגוד לאופרטור A-- שמחזיר את התמונה A **בעצמה**, כפי שהיא **לאחר** השינוי).
- A(x,y) – אופרטור סוגריים, כאשר x ו-y, הם מסוג של unsigned int, והוא **מחזיר הפניה (reference)** לתא במקום ה-(x,y) של התמונה. התא ה-(0,0), כלומר A(0,0), זה התא השמאלי העליון של התמונה, והתא ה־(Wa-1,Ha-1), כלומר A(Wa-1,Ha-1), יהיה תא הימני התחתון של התמונה.
  הערה חשובה: האופרטור הזה משמש לקריאה ולכתיבה, כי על ידי זה שחוזרת הפניה (reference) לתא המבוקש, נוכל לשנות את התאים בתמונה כפי רצוננו.
- עוד A(x,y) – עוד אופרטור סוגריים, כאשר x ו-y, הם מסוג של unsigned int, והוא **מחזיר הפניה קבועה (const reference)** לתא במקום ה-(x,y) של התמונה. כפי שהוסבר לעיל. האופרטור הזה משמש לקריאה בלבד, מכיוון שהערך המוחזר הוא למעשה const אז מובטח לנו (עד כדי const_cast) שהתמונה A לא תשתנה.

הערות ורמזים למימוש האופרטורים:

שימו לב האם אתם יכולים, ואף נדרשים, להשתמש במימוש של אופרטורים מסוימים באופרטורים אחרים (מאותה מחלקה או ממחלקה אחרת) שכבר מימשתם, כדי למנוע קוד כפול.

שימו לב שישנם אופרטורים שלא משנים את התמונה (כמו למשל האופרטור חיבור) לכן אתם חייבים להצהיר עליו שהוא אופרטור const. בכל האופרטורים תחשבו אם הם const או לא.

פונקציות כלליות שאתם נדרשים לממש:

במחלקת Image, בנוסף לכל האופרטורים שהגדרנו, אתם צריכים לממש את הפונקציות הציבוריות GetHeight ו-GetWidth. חוץ מהן, למחלקת Image לא יהיו יותר פונקציות ציבוריות.
בשאר המחלקות אתם רשאים להוסיף ולהגדיר פונקציות ציבוריות ואופרטורים ציבוריים כפי הצורך.

קובץ ה־README:

יש לכלול קובץ README שיקרא README.doc, README.docx או README.txt (ולא בשם אחר).
הקובץ יכול להיכתב בעברית ובלבד שיכיל את הסעיפים הנדרשים.

קובץ זה יכיל לכל הפחות:

1. כותרת.
2. פרטי הסטודנט: שם מלא כפי שהוא מופיע ברשימות המכללה, ת"ז.
3. הסבר כללי של התרגיל.
4. **תיכון (design): הסבר קצר מהם האובייקטים השונים בתוכנית, מה התפקיד של כל אחד מהם וחלוקת האחריות ביניהם ואיך מתבצעת האינטראקציה בין האובייקטים השונים.**
5. רשימה של הקבצים שנוצרו ע"י הסטודנט, עם הסבר קצר (לרוב לא יותר משורה או שתיים) לגבי תפקיד הקובץ.
6. מבני נתונים עיקריים ותפקידיהם.
7. אלגוריתמים הראויים לציון.
8. באגים ידועים.
9. הערות אחרות.

יש לתמצת ככל שניתן אך לא לוותר על אף חלק. אם אין מה להגיד בנושא מסוים יש להשאיר את הכותרת ומתחתיו פסקה ריקה. תכתבו ב־README כל דבר שרצוי שהבודק ידע כשהוא בודק את התרגיל.

אופן ההגשה:

הקובץ להגשה: יש לדחוס כל קובץ הקשור לתרגיל, למעט מה שיצוין להלן, לקובץ ששמו exN_name.zip, כאשר N הוא מספר התרגיל ו־name הוא השם המלא. במקרה של הגשה בזוג, שם הקובץ יהיה לפי התבנית exN_name1_name2.zip, עם שמות המגישים בהתאמה (ללא רווחים; גם בשמות עצמם יש להחליף רווחים בקו תחתי או להצמיד את שני חלקי השם).

לפני דחיסת תיקיית ה־Solution שלכם יש למחוק את הפריטים הבאים:

- תיקיות בשם debug או release (לרוב יש יותר מתיקייה אחת בשם זה, אחת בתיקיית ה־Solution ואחת בתיקיית כל פרויקט).

- תיקייה בשם x64, אם קיימת.
- תיקייה בשם vs. שאמורה להיות בתיקייה עם ה־Solution.

זכרו שגם את הקבצים שנתנו לכם אתם צריכים לצרף, והם צריכים להיות ללא שינוי, כפי שקיבלתם אותם.

ככלל אצבע, אם קובץ ה־zip שוקל יותר ממ"ב אחד או שניים, כנראה שלא מחקתם חלק מהקבצים הבינאריים המוזכרים.

את הקובץ יש להעלות ל־Moodle של הקורס למשימה המתאימה.

הגשה חוזרת: אם מסיבה כלשהי סטודנט מחליט להגיש הגשה חוזרת יש לוודא ששם הקובץ זהה לחלוטין לשם הקובץ המקורי. אחרת, אין הבודק אחראי לבדוק את הקובץ האחרון שיוגש.

כל שינוי ממה שמוגדר פה לגבי צורת ההגשה ומבנה ה־README עלול לגרור הורדת נקודות בציון.

מספר הערות:

1. שימו לב לשם הקובץ שאכן יכלול את שמות המגישים.

2. שימו לב שעליכם לשלוח את תיקיית ה־solution כולה, לא רק את קובצי הקוד שיצרתם. עקבו אחרי ההסבר המפורט באתר, במקרה שאתם לא בטוחים איך למצוא את התיקייה. תרגיל שלא יכלול את ה־solution, לא יתקבל וידרוש הגשה חוזרת (עם כללי האיחור הרגילים).

המלצה כללית: אחרי שהכנתם את הקובץ להגשה, העתיקו אותו לתיקייה חדשה, חלצו את הקבצים שבתוכו ובדקו אם אתם מצליחים לפתוח את ה־solution שבתוכו ולקמפל את הקוד. הרבה טעויות של שכחת קבצים יכולות להימנע על ידי בדיקה כזו.

<u>בהצלחה!</u>