מכללת הדסה, החוג למדעי המחשב
מבוא לתכנות מונחה עצמים והנדסת תוכנה
סמסטר א', תשפ"א

תרגיל 3

תאריך אחרון להגשה:

הנביאים – יום א', 06/12/20, בשעה 23:59

מטרת 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) פונקציית הריסה (destructor) ובהעמסת אופרטורים (operator overloading).

תיאור כללי:

בתרגיל זה נבנה מחלקת Image – מחלקה שתייצג תמונה בגווני שחור/לבן/אפור בלבד. התמונה תהיה בגובה H וברוחב W כלשהם, כאשר H ו־W מתקבלים בבנאי. התמונה תורכב מפיקסלים כפי שיפורט בהמשך.

מכיוון שאחת ממטרות התרגיל היא תרגול של שימוש נכון בהקצאות זיכרון דינמיות, ובפרט הניהול שלהן בזמן העתקה, השמה והריסה, אין להשתמש בתרגיל זה במחלקות מוכנות (כגון std::vector, std::list או std::string) שתעקופנה את הצורך שלכם ליצור את הפונקציונליות הזו בעצמכם.

אתם נדרשים לחשוב על שיקולים של תיכון – 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. כמו כן, חלוקה נכונה של התפקידים בין מחלקות שונות היא מפתח לתיכון ראוי, וגם תקל בסופו של דבר על המימוש.

כדי להגיע לתיכון ראוי, אתם מתבקשים להגדיר מחלקות נוספות מלבד המחלקה העיקרית שהוגדרה כאן, מחלקת Image, מחלקות עזר שבהן תשתמשו כדי לממש בסופו של דבר את מחלקת Image. כהנחיה ראשונית לתיכון נציין שצריך ליצור מחלקה שתייצג פיקסל - תא אחד בתמונה. מחלקה נוספת תהיה מחלקה שתנהל את מבנה הנתונים שתבחרו ליצור. במחלקת Image תישאר האחריות רק למימוש הלוגיקה של התמונות, והיא תהיה פטורה מלהתעסק בניהול זיכרון.

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

בנוסף, הדרישות כוללות כמעט אך ורק מחלקות, אבל עדיין מומלץ מאוד שתכינו לעצמכם בפונקציית ה־main (מלבד הדרישה הספציפית שיש לגביה) מספר בדיקות לוודא שהמחלקות השונות עובדות כראוי. בנוסף, אם תבחרו להשתמש בקבצים שהעלנו, תמצאו בהם, בתיקיית tests, מספר בדיקות למחלקות Pixel ו־Image. אתם יכולים להיעזר בהן לבדיקת הקוד שלכם ואולי גם תרצו להוסיף בדיקות נוספות משלכם לכיסוי טוב יותר של הקוד. בשלב הראשון אולי תרצו לשים בהערה חלקים של הבדיקות האלה או להוריד את שמות הקבצים שלהן מקובץ ה־CMakeLists.txt שבתיקייה, כדי למנוע שגיאות צפויות. כדי להריץ את הטסטים, אפשר פשוט לבחור את Tests.exe במקום oop1_ex03.exe ולהריץ.

פירוט הדרישות:

מחלקת Pixel:

המחלקה תייצג 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).

פונקציות למחלקת Pixel:

Pixel(unsigned char pixel = ' ') - בנאי המקבל משתנה מסוג unsigned char. בנאי זה משמש גם כבנאי שממיר unsigned char לפיקסל (הוא לא explicit) וגם כבנאי ברירת מחדל (לצבע לבן, בגלל ערך ברירת המחדל שמוגדר לפרמטר).

עבור הגדרות האופרטורים שלהלן, נניח ש- P1 ו-P2 הם שני פיקסלים.

אופרטורים בינאריים שאתם נדרשים לממש (להעמיס):

- P1==P1, P1!=P2 – השוואת פיקסלים. הפיקסלים שווים אם ורק אם שווים בצבעם.
- <<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 - החתימה של האופרטור הזה היא:

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

בקובץ ה־header, כדי להצהיר על הפונקציה, צריך להוסיף #include <iosfwd>, ולצורך המימוש, בקובץ ה־cpp, צריך להוסיף #include <ostream>.

חיתוך ואיחוד:

- 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.

מחלקת מבנה הנתונים:

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

מחלקת Image:

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.

Image(int height, int width, unsigned char pixel) - בנאי שיאתחל את התמונה לגובה ולרוחב הנתונים. בכל התאים, כלומר הפיקסלים, הצבע יהיה ע"פ הפרמטר pixel.

Image(const Image& other) - בנאי העתקה, יעתיק את התמונה המתקבלת לתוך האובייקט החדש. חשוב לציין שחייבת להיות כאן העתקה עמוקה, כך ששינוי באובייקט אחד לא ישפיע כלל על האובייקט השני.

עבור הגדרות האופרטורים שלהלן נניח שקיימות 2 תמונות A ו־B. תמונה A בגובה Ha וברוחב Wa, ותמונה B בגובה Hb וברוחב Wb.

אופרטורים בינאריים שאתם נדרשים לממש (להעמיס):

- A==B, A!=B – השוואת תמונות. התמונות שוות אם ורק אם הגובה, הרוחב וכל הפיקסלים של התמונות שווים.

- 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 תהיה לכם דליפת זיכרון. כפי שהזכרנו בבנאי העתקה: חשוב לציין שחייבת להיות כאן העתקה עמוקה, כך ששינוי באובייקט אחד לא ישפיע כלל על האובייקט השני.

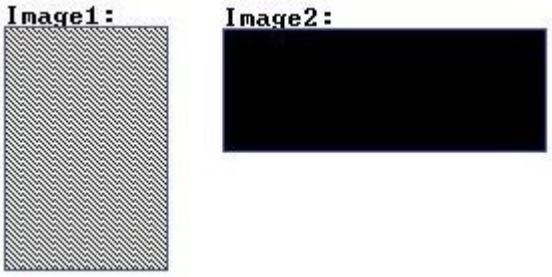
נא לשים לב: אופרטור ההשמה דומה במימושו לבנאי העתקה. תוודאו שאין לכם קוד כפול.

- 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.

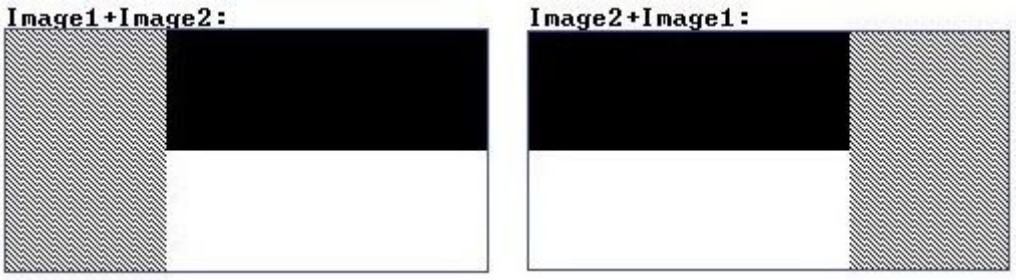
נא לשים לב: התמונות A ו־B אינן משתנות כלל. האופרטור מחזיר את התוצאה, אבל לא משנה כלל את התמונות עצמן.

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.

דוגמה לחיבור של תמונות. בהתייחס לשתי התמונות הללו:



תוצאות החיבור שלהן תהיה (שימו לב לחוסר הקומוטטיביות):



- 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).

. תזכורת - החתימה של האופרטור הזה היא:

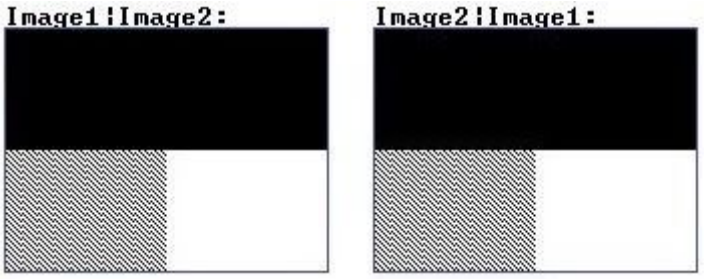
```
std::ostream& operator<<(std::ostream&, const Image&);
```

- 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.

דוגמה לאיחוד של תמונות (בהתייחס לתמונות שלעיל. הפעם שימו לב לקומוטטיביות):



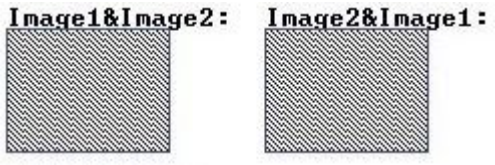
- A|=B – איחוד והשמה של התמונה. פעולה זה שקולה לפעולה: A=A|B.

- 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.

דוגמה לחיתוך (שוב, בהתייחס לתמונות שלעיל):



- A&=B – חיתוך והשמה של התמונה. פעולה זה שקולה לפעולה: A=A&B.

- 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, תוחזר תמונה ריקה.

- n*A – כפל בסקלר משמאל, מתנהג בדיוק כמו כפל בסקלר מימין.

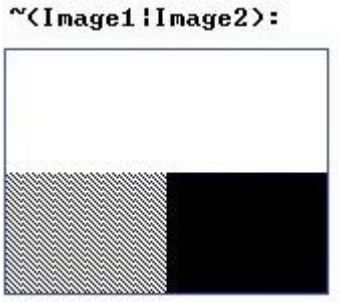
נא לשים לב: בכפל בסקלר מימין ומשמאל, התמונה A אינה משתנה כלל. האופרטור מחזיר את התוצאה, אבל לא משנה כלל את התמונה עצמה.

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

אופרטורים אונריים שאתם נדרשים לממש:

- ~A – תשליל. התוצאה תהיה תמונה בגודל זהה, כאשר פיקסלים לבנים - יהפכו לשחורים, פיקסלים שחורים - יהפכו ללבנים ופיקסלים אפורים יישארו כמות שהם, אפורים. שימו לב - האופרטור מחזיר את התוצאה, אך לא משנה את התמונה עצמה.

דוגמה לתשליל (במקרה הזה, תשליל של איחוד התמונות):



- 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.

פונקציות ציבוריות נוספות שאתם צריכים לממש:

GetHeight ו-GetWidth - מחזירות את הגובה והרוחב בהתאמה.

מלבד מה שפורט לעיל, למחלקת Image לא יהיו פונקציות ציבוריות נוספות.

בשאר המחלקות אתם רשאים להוסיף ולהגדיר פונקציות ציבוריות ואופרטורים ציבוריים לפי הצורך.

הערות למימוש האופרטורים:

- שימו לב האם אתם יכולים להשתמש במימוש של אופרטורים מסוימים באופרטורים אחרים שכבר מימשתם. הרבה מהאופרטורים שהוזכרו לעיל, יכולים להיות ממומשים בשורה בודדת, בעזרת שימוש באופרטורים האחרים שכבר הגדרתם.
- במקרים מסוימים אולי תרצו להעמיס אופרטורים נוספים שלא הוזכרו כאן בפירוש, כדי לפשט מימוש של האופרטורים המוזכרים. אם תעשו זאת, האופרטורים האלה צריכים להיות גם הם ציבוריים (בניגוד לפונקציות עזר פנימיות שאולי תגדירו, שעליהן להיות פרטיות, כרגיל).
- שקלו היטב האם לממש אופרטור מסוים כפונקציית מחלקה (member function) או כפונקציה גלובלית. אם אין הכרח לממש בפונקציית מחלקה, פונקציה גלובלית עדיפה. שימו לב גם מה טיפוס ההחזרה הנכון (by value או by reference, למשל) וכן אלו פונקציות מחלקה צריכות להיות מוגדרות כ־const.
- למרות שהתרגיל עוסק רבות באופרטורים, שימו לב שאם אתם כרגע רק מתחילים לעסוק בנושא אופרטורים ומסובך לכם להתחיל לעבוד ישירות על העמסת אופרטורים, ניתן לכתוב את רוב הקוד בעזרת הגדרת פונקציות רגילות במקום אופרטורים, וכשתרגישו מספיק בטוחים עם אופרטורים תוכלו להחליף את הגדרת הפונקציה בהגדרה המתאימה לאופרטור.

תיעוד המחלקות:

אין צורך לתעד קוד שהוא מובן מאליו (למשל בגלל שמות משמעותיים או בגלל שהפונקציות הוגדרו כך בהנחיות התרגיל), זה מיותר גם בהצהרות וגם במימושים. רק אם מדובר ברכיב ייחודי בלתי מובן או בפונקציות שהן פחות טריוויאליות (לא setToDefault, נניח) יש להוסיף תיעוד בקוד.

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 ומתחתיה 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_firstname_lastname.zip, כאשר N הוא מספר התרגיל ו־firstname_lastname הוא השם המלא (לדוגמא אלברט איינשטיין יגיש כך את התרגיל הראשון: ex1_albert_einstein.zip). במקרה של הגשה בזוג, שם הקובץ יהיה לפי התבנית exN_firstname1_lastname1_firstname2_lastname2.zip, עם שמות המגישים בהתאמה (ללא רווחים; כלומר, גם בשמות עצמם יש להחליף רווחים בקו תחתי, כפי המודגם לעיל). כמו כן, במקרה של הגשה בזוג, **רק אחד** מהמגישים יגיש את הקובץ ולא שניהם.

לפני דחיסת תיקיית הפרויקט שלכם יש למחוק את הפריטים הבאים:

- תיקייה בשם out, אם קיימת
- תיקייה בשם vs.

שתי התיקיות האלה נמצאות בתיקייה הראשית (זו שאנחנו פותחים בעזרת VS). התיקייה vs. לפעמים מוסתרת, אבל אם תפתחו את קובץ ה־zip שיצרתם, בוודאי תוכלו למצוא אותה ולמחוק אותה.
ככלל אצבע, אם קובץ ה־zip שוקל יותר ממ"ב אחד או שניים, כנראה שלא מחקתם חלק מהקבצים הבינאריים המוזכרים.

וודאו כי קובץ ה־zip מכיל תיקייה ראשית אחת, ורק בתוכה יהיו כל הקבצים ותתי התיקיות של הפרויקט.

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 הפרוייקט כולה, לא רק את קובצי הקוד שיצרתם. תרגיל שלא יכלול את כל הקבצים הנדרשים, 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 התיקייה הזו ולקמפל את הקוד. הרבה טעויות של שכחת קבצים יכולות להימנע על ידי בדיקה כזו.

בהצלחה!