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

תרגיל 3

תאריך אחרון להגשה:

הנביאים – יום א', 11/12/2022, בשעה 23:59

שטראוס גברים – יום א', 11/12/2022, בשעה 23:59

שטראוס נשים – מוצאי שבת, 10/12/2022, 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) והורס (Destructor) ובהעמסת אופרטורים (Operator Overloading).

תיאור כללי:

בתרגיל זה נבנה מחלקת Vector – מחלקה שתייצג וקטורים (במובן המתמטי) מעל החוג $\mathbb{Z}[i]$ (חוג השלמים של גאוס: מספרים מהצורה a+bi, כמו המרוכבים, אבל a,b שניהם מספרים שלמים, לא ממשיים כלשהם) ותאפשר פעולות על וקטורים כאלה.

כדי להגיע לתיכון ראוי, הפתרון צריך לכלול מחלקות נוספות מלבד מחלקת Vector, מחלקות עזר שישמשו כדי לממש בסופו של דבר את מחלקת Vector. לשם כך, ניצור מחלקה שתייצג מספרים בחוג $\mathbb{Z}[i]$. מחלקה נוספת תהיה מחלקה שמממשת מבנה נתונים כלשהו (ראו בהמשך) וכחלק מכך אחראית לניהול הזיכרון עבור מבנה הנתונים הזה (הקצאות, שחרורים וכו'). במחלקת Vector תישאר האחריות רק למימוש ההתנהגות המתמטית של הווקטורים.

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 או std::list) שתעקופנה את הצורך שלכם ליצור את הפונקציונליות הזו בעצמכם.

בפתרון, יש להתחשב בשיקולים של תיכון – עיצוב מונחה עצמים (OOD), לא פחות ואולי אף יותר מאשר בשיקולים של יעילות. למשל, למימוש רוב האופרטורים המפורטים בהמשך, 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.

עוד הערה חשובה: אמנם רוב הגדרת התרגיל עוסקת בהגדרת האופרטורים הנדרשים, 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 מספר בדיקות לוודא שהמחלקות השונות עובדות כראוי. בנוסף, אם תבחרו להשתמש בקבצים החדשים שהעלנו, תמצאו בהם, בתיקיית test, מספר בדיקות למחלקות Zi ו־Vector. ניתן להיעזר בהן לבדיקת הקוד שלכם ואולי גם תרצו להוסיף בדיקות נוספות לכיסוי טוב יותר של הקוד. בשלב הראשון אולי תרצו לשים בהערה חלקים של הבדיקות האלה או לשנות את שמות הקבצים שלהן כך שלא יסתיימו בסיומת cpp., כדי למנוע שגיאות צפויות כל עוד המחלקות הרלוונטיות עדיין לא קיימות. כדי להריץ את הטסטים, אפשר לבחור את tests.exe במקום oop1_ex03.exe ולהריץ.

פירוט הדרישות:

מחלקת Zi:

המחלקה תייצג מספר בחוג $\mathbb{Z}[i]$, כפי שהוסבר.

פונקציות למחלקת Zi:

Zi(int a = 0, int b = 0) – בנאי המקבל משתנים מסוג int. הבנאי הזה משמש גם כבנאי שממיר int למספר ב־$\mathbb{Z}[i]$ (הוא לא explicit) וגם כבנאי ברירת מחדל (לערך 0, בגלל ערך ברירת המחדל שמוגדר לפרמטר).

הפונקציות הציבוריות שתממשו הן:

int real() const – מחזירה את החלק הממשי של המספר (ה־a).

int imag() const – מחזירה את החלק המרוכב של המספר (ה־b).

int norm() const – מחזירה את הנורמה של המספר, המוגדרת כ־$a^2 + b^2$.

Zi conj() const – מחזירה את הצמוד (conjugate) של המספר, כלומר אותו חלק ממשי ומינוס של החלק המדומה.

bool dividedBy(const Zi& divisor) const – מחזירה האם המספר הנוכחי מתחלק ב־divisor ללא שארית. ניתן לבדוק זאת בעזרת חישוב השארית (כפי שמוסבר בהמשך), או באופן ישיר יותר על ידי הכפלה של המספר בצמוד של המחלק, ובדיקה האם המספרים שקיבלנו מתחלקים ללא שארית בנורמה של המחלק (דומה לתחילת אלגוריתם החילוק המוסבר בהמשך).

למחלקה זו אתם מתבקשים להעמיס את האופרטורים הבאים:

אופרטורים מתמטיים – חיבור, חיסור, כפל, חילוק ושארית (מודולו, %) בשתי הצורות שלהם, עם סימן השוויון ובלעדיו. המשמעות של חיבור וחיסור ברורה. גם המשמעות של כפל צריכה להיות ברורה מההיכרות שלנו עם מספרים מרוכבים (פתיחת סוגריים וכינוס איברים, עם התזכורת ש־$i^2 =- 1$).

החילוק והשארית מוגדרים באופן הבא: נכפיל את המחלק בצמוד של המחלק, נחלק בנורמה של המחלק, ונעגל את שני המספרים שקיבלנו לשלם הקרוב ביותר כדי לקבל את המנה.

לדוגמה: $\frac{27-23i}{8+i} = \frac{(27-23i)(8-i)}{65} = \frac{193-211i}{65}$ ומכיוון ש־$\frac{193}{65} = 2.969\ldots \quad - \frac{211}{65} =- 3.246\ldots$, נעגל לשלם הקרוב ונקבל שהמנה היא $3 - 3i$ והשארית (שברור איך לחשב אותה אחרי שיש לנו מחולק, מחלק ומנה) תהיה $0 - 2i$

אופרטור מינוס אונרי (Unary operator, אופרטור חד מקומי, שעובד על פרמטר אחד) – המשמעות ברורה.

אופרטורים לוגיים (אופרטורי השוואה) – נממש את האופרטורים שווה ושונה.

אופרטור הדפסה (אופרטור שליפה, Extraction operator, <<) – מדפיס ל־ostream את המספר הנתון. תזכורת: החתימה של האופרטור הזה היא

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

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

מחלקת מבנה הנתונים:

כפי שהוסבר לעיל, מחלקה זו מיועדת לנהל מבנה נתונים כלשהו בזיכרון. עליכם להחליט מה סוג מבנה הנתונים שנכון לממש כאן. תנו למחלקה זו שם מתאים המייצג את המשמעות שלה, לפי מבנה הנתונים שבחרתם לממש (לא לקרוא לה DataStructure!). המטרה העיקרית שלה היא לטפל בנושאי ניהול הזיכרון.

מכיוון שעדיין לא למדנו את הטכניקה הנדרשת, לא נוכל לממש מבנה נתונים גנרי לחלוטין שיכול להחזיק כל טיפוס נתונים שנרצה (כלומר, לא נוכל כעת לממש משהו כמו std::vector). לכן, מבנה הנתונים שלנו ישמור אובייקטים מסוג Zi.

שימו לב שאנחנו מבקשים להגדיר מספר פונקציות ואופרטורים במחלקה זו. זה לא בהכרח אומר שמחלקת Vector חייבת להשתמש בכולם בדרך כזו או אחרת! חלקם נמצאים כאן בעיקר במטרה לתרגל נקודות נוספות בנוגע לניהול הזיכרון או כדי להדגים לנו שיכולים להיות אופרטורים דומים שמבצעים פעולות שונות לחלוטין עבור מחלקות שונות.

דרישות למחלקה זו:

במחלקה זו נגדיר בנאי המקבל את גודל מבנה הנתונים המבוקש וערך (Zi) שאליו יאותחלו כל איברי מבנה הנתונים. בנאי זה ישמש גם כבנאי ברירת המחדל. ברירות המחדל של הבנאי יהיו גודל 0 וערך ברירת המחדל של Zi. כלומר, הפרמטרים של הבנאי יהיו (int size = 0, const Zi& value = Zi()). על הבנאי הזה להיות explicit כדי למנוע המרה לא רצויה.

בנאי נוסף במחלקה יקבל מערך של Zi ואת הגודל שלו (כ־int) ויבנה מבנה נתונים בגודל הזה שיהיה מאותחל בעזרת האיברים שבמערך. כלומר, הפרמטרים של הבנאי יהיו (const Zi arr[], int size).

כצפוי, במחלקה זו נידרש לטפל בהעתקה (כלומר, בנאי העתקה ואופרטור השמה) והריסה כראוי.

אופרטורים שנגדיר במחלקה זו יהיו:

אופרטור החיבור (בשתי הצורות, רק חיבור וחיבור יחד עם השמה) שישרשר את שני מבני הנתונים המתקבלים למבנה נתונים אחד.

אופרטור [ ] (לגישה כמו במערך) שיקבל אינדקס כ־int ויחזיר הפניה לאיבר במיקום הזה במבנה הנתונים. חשוב לזכור להגדיר את שתי הגרסאות של האופרטור, זו שמשמשת לקריאה בלבד (const) וזו שמשמשת גם לכתיבה. בדומה למקובל בשפה ובספרייה הסטנדרטית בהקשרים דומים, אין דרישה לוודא שהאינדקס תקין (האחריות על הבדיקה הזו מוטלת על הקוד שקורא לפונקציה).

אופרטורי שווה ושונה. שני מבני הנתונים שווים רק אם הם באותו הגודל, ומכילים איברים שווים ובאותו הסדר.

int size() const – מחזירה את גודל מבנה הנתונים.

bool empty() const – מחזירה האם מבנה הנתונים ריק.

מחלקת Vector:

גם כאן יהיו שני בנאים שדומים למקביליהם במבנה הנתונים:

explicit Vector (int size = 0, const Zi& init = {}) – מקבל גודל וערך לאתחול, ומשמש גם כבנאי ברירת מחדל אבל לא יכול לשמש להמרה אוטומטית (הוא explicit).

Vector (const Zi arr[ ], int size) – מקבל מערך ואת גודל המערך ומאתחל בעזרתם.

(שאלה למחשבה: האם צריך לטפל כאן "ידנית" בהעתקה ובהריסה?)

אופרטורים שנגדיר כאן:

חיבור וחיסור – במשמעות הרגילה של חיבור וחיסור וקטורים, עושים את החיבור או החיסור איבר־איבר.

כפל וקטורים – גם זה נבצע ככפל נקודתי, כלומר, כפל איבר־איבר.

כפל של וקטור בסקלר (אובייקט מסוג Zi) – במשמעות הרגילה, מכפיל את כל איברי הווקטור בסקלר הנתון. שימו לב לאפשר כפל גם מימין וגם משמאל.

כרגיל, בחיבור, בחיסור ובכפל, נגדיר גם את הגרסה עם ההשמה (שאלה למחשבה: האם גרסת ההשמה רלוונטית גם לכפל בסקלר? בכל צורות הכפל?).

שימו לב שייתכן שנקבל באופרטורים האלה כאופרנדים שני וקטורים בגודל שונה. במקרה כזה, נבצע את הפעולה על פי המימד של הווקטור הגדול מבין השניים, והאיברים ה"חסרים" בווקטור הקטן יותר ייחשבו כ־0.

נגדיר גם את האופרטור האונרי של מינוס. הוא מחזיר עותק שמכיל אותם האיברים כמו במקור, אחרי פעולת מינוס. (למשל, אם המקור היה עם האיברים 5+0i 3-6i, התוצאה תהיה וקטור עם האיברים -5+0i -3+6i)

גם פה נגדיר את האופרטורים שווה ושונה, ושוב, במשמעות הרגילה – וקטורים שווים רק אם הם באותו הגודל, ומכילים איברים שווים ובאותו הסדר.

כמו כן, נגדיר אופרטור גישה (אופרטור סוגריים מרובעים) שיקבל אינדקס כ־int ויחזיר הפניה לאיבר במיקום הזה בווקטור. גם כאן חשוב לזכור להגדיר את שתי הגרסאות של האופרטור, זו המשמשת לקריאה בלבד (const) וזו שמשמשת גם לכתיבה.

int size() const – מחזירה את גודל הווקטור.

לסיום, נגדיר גם פה אופרטור הדפסה. הוא ידפיס את איברי הווקטור לפי הסדר עם רווחים ביניהם. ושוב נזכיר שהחתימה היא:

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

הערות למימוש האופרטורים:

- שימו לב האם אפשר להשתמש במימוש של אופרטורים מסוימים באופרטורים אחרים שכבר מומשו. הרבה מהאופרטורים שהוזכרו לעיל, יכולים להיות ממומשים בשורה בודדת, בעזרת שימוש באופרטורים האחרים שכבר הגדרתם.
- כאמור, פשטות המימוש יכולה להכריע יותר מאשר ההתלבטות על מספר ההעתקות שאולי קורה וכדומה. במיוחד שימושי לשקול את זה בהחלטות מתי לממש אופרטור בעזרת הצורה המשולבת בהשמה (למשל, + בעזרת =+) ומתי להיפך.

- 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 (או בכלל מחוץ למחלקה), 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. בשום מקרה אין להשתמש ב־friend.
- 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.
- כדי לעגל מספר לשלם הקרוב, ניתן להיעזר בפונקציית ()std::round מספריית cmath. (שימו לב, floor או trunc או סתם המרה ל־int לא יתנו את התוצאה הרצויה!)
- נזכיר שכדי להדפיס את הסימן של מספר, גם אם הוא חיובי, ניתן "להדפיס" לפניו את std::showpos. שימו לב שההגדרה של std::showpos איננה חד פעמית ותשפיע על ההדפסה ל־stream הזה גם בפעמים הבאות, לכן חשוב "לכבות" אותו לאחר ההדפסה (בעזרת std::noshowpos).

תיעוד המחלקות:

אין צורך לתעד קוד שהוא מובן מאליו (למשל, בגלל שמות משמעותיים או אופרטור שמשמעותו ברורה), זה מיותר גם בהצהרות וגם במימושים. יש להוסיף תיעוד בקוד במקרים הבאים:

- תיעוד כללי (וקצר) על ממשק המחלקה (מטרת המחלקה עצמה ופונקציות ציבוריות למיניהן)
- בפונקציות פחות טריוויאליות (לא setToDefault, נניח)
- כשיש לוגיקה לא טריוויאלית (במימוש)
- במקרים של קוד לא ברור מאליו ו"מפתיע" (למשל, העברת ערך לא חוקי בכוונה כדי לקבל כתוצאה את ברירת המחדל)

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. רשימה של הקבצים שיצרנו, עם הסבר קצר (לרוב לא יותר משורה או שתיים) לגבי תפקיד הקובץ.
5. מבני נתונים עיקריים ותפקידיהם.
6. אלגוריתמים הראויים לציון.
7. תיכון (design): הסבר קצר מהם האובייקטים השונים בתוכנית, מה התפקיד של כל אחד מהם וחלוקת האחריות ביניהם ואיך מתבצעת האינטראקציה בין האובייקטים השונים.
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. **נכתוב ב־README כל דבר שרצוי שהבודק ידע כשהוא בודק את התרגיל.**

אופן ההגשה:

הקובץ להגשה: ניתן ליצור בקלות קובץ zip המותאם להגדרות ההגשה המפורטות להלן ישירות מתוך VS, כפי המוסבר תחת הכותרת "יצירת קובץ ZIP להגשה או לגיבוי" בקובץ "הנחיות לשימוש ב־Visual Studio 2022". אנא השתמשו בדרך זו (אחרי שהגדרתם כראוי את שמות הצוות ב־MY_AUTHORS: **נשים את שמות המגישים בתוך המרכאות, נקפיד להפריד בין השם הפרטי ושם המשפחה בעזרת קו תחתי ואם יש יותר ממגיש אחד, נפריד בין השמות השונים בעזרת מקף (מינוס '-')**) וכך תקבלו אוטומטית קובץ zip המותאם להוראות, בלי טעויות שיגררו אחר כך בעיות בבדיקה.

באופן כללי, הדרישה היא ליצור קובץ zip בשם oop1_ex0N-firstname_lastname.zip (או במקרה של הגשה בזוג – oop1_ex0N-firstname1_lastname1-firstname2_lastname2.zip), כשהקובץ כולל את כל קובצי הפרויקט, למעט תיקיות out ו־vs. כל הקבצים יהיו בתוך תיקייה ראשית אחת.

את הקובץ יש להעלות ל־Moodle של הקורס למשימה המתאימה. בכל מקרה, **רק אחד** מהמגישים יגיש את הקובץ ולא שניהם.

הגשה חוזרת: אם מסיבה כלשהי החלטתם 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. נשים לב לשם הקובץ שאכן יכלול את שמות המגישים.

2. נשים לב לשלוח את תיקיית הפרוייקט כולה, לא רק את קובצי הקוד שהוספנו. תרגיל שלא יכלול את כל הקבצים הנדרשים, לא יתקבל וידרוש הגשה חוזרת (עם כללי האיחור הרגילים).

המלצה כללית: אחרי הכנת הקובץ להגשה, נעתיק אותו לתיקייה חדשה, נחלץ את הקבצים שבתוכו ונבדוק אם ניתן לפתוח את התיקייה הזו ולקמפל את הקוד. הרבה טעויות של שכחת קבצים יכולות להימנע על ידי בדיקה כזו.

בהצלחה!