מכללת הדסה, החוג למדעי המחשב
תכנות מונחה עצמים ופיתוח משחקים
סמסטר ב', תשפ"ב

תרגיל 1

תאריך אחרון להגשה:

הנביאים – יום א', 13/03/2022, בשעה 23:59

שטראוס-גברים – יום ה', 10/03/2022, בשעה 23:59

שטראוס-נשים – מוצאי שבת, 12/03/2022, בשעה 23:59

מטרת התרגיל:

ריענון של העקרונות שנלמדו בסמסטר א' וחזרה עליהם, בפרט תכנון ממשק ותיכון, העמסת אופרטורים, ירושה ופולימורפיזם וכן שימוש בכלים מהספרייה הסטנדרטית, כולל מצביעים חכמים.

תיאור כללי:

בתרגיל זה נממש "מחשבון קבוצות" ניתן לתכנות. המחשבון מחשב את תוצאות הפעולות המבוקשות על קבוצות (מתמטיות, ללא כפילויות) של מספרים שלמים לפי הקלט מהמשתמש. המשתמש יכול גם ליצור פעולות מורכבות יותר ולהוסיף אותן למחשבון ואחר כך לבקש חישוב של תוצאה של הפעלתן על קבוצות מסוימות. לשם הפשטות, בתרגיל זה נחזור לעבוד בטרמינל.

פירוט הדרישות:

המחשבון מחזיק רשימת פעולות על קבוצות שאותן הוא מסוגל לבצע. שלוש הפעולות שצריכות להיות קיימות כבר בהפעלת המחשבון הן:

- A U B – איחוד של הקבוצות A ו־B – כמו שניתן לראות, נסמן את האיחוד ב־U (האות האנגלית ב־Uppercase), שכמעט זהה לסימון המקובל של איחוד – ∪
- A ^ B – חיתוך הקבוצות A ו־B – את החיתוך סימנו ב־^ (Shift+6), כי זה הסימן הכי קרוב במקלדת לסימון המקובל של חיתוך – ∩
- A - B – ההפרש בין הקבוצות A ו־B, כלומר האיברים שנשארים ב־A אחרי החסרת האיברים שנמצאים ב־B (כלומר, הפרש רגיל, לא הפרש סימטרי) – נסמן אותו פשוט כחיסור (סימן מינוס)

מהלך ריצת התוכנית:

תחילה תודפס למסך רשימת הפעולות (שכזכור, בתחילת התוכנית מכילה את הרשימה שהוזכרה לעיל). לכל פעולה ברשימה יש מספר סידורי שמודפס לידה בהדפסת רשימת הפעולות, ובעזרתו המשתמש יכול להתייחס אליה בפקודות שהוא מקליד. המספרים תמיד רציפים, כלומר גם אם היו פעולות שנמחקו מהרשימה לא יהיו "חורים" ודילוגים במספור. עבור כל פעולה, יופיעו המספר

הסידורי שלה ונוסחת הפעולה (ראו דוגמאות בהמשך). אחרי הרשימה, תופיע בקשה להכנסת פקודה, עם תזכורת לגבי הדרך לקבל עזרה על הפקודות האפשריות.

התכנית תמתין לקבלת פקודה מהמשתמש, תבדוק את הקלט הנתון, ואם מתאפשר – תבצע את הפעולה תוך הדפסת הפלט הרלוונטי. אם הפקודה לא חוקית או הפרמטרים עבור הפקודה שגויים, תודפס הודעת שגיאה.

בכל מקרה, בין אם הפקודה התבצעה ובין אם הצגנו שגיאה עבורה, כאן נחזור לראש הלולאה, כלומר שוב תודפס רשימת הפעולות (שאולי השתנתה בינתיים, אם המשתמש הוסיף או מחק פעולה כלשהי) וכן הלאה.

רשימת הפקודות האפשריות:

הערה: הפקודה שהמשתמש מקליד היא תחילת המילה. בסוגריים מוצגת השלמת המילה המלאה כדי לעזור בהבנת המשמעות. גם במסך העזרה נדפיס כמו שמופיע פה, עם סוגריים על השלמת המילה.

... eval(uate) num – מחשבת את הערך של הפעולה שמספרה num, עבור הקלט שניתן בשאר הארגומנטים (שמיוצגים כאן בטקסט העזרה כ־...). הקלט צריך להיות קבוצות לפי הסדר שלהן בפקודה. כל קבוצה מורכבת תחילה ממספר האיברים לקריאה מהקלט, ואז המספרים שממנה היא מורכבת. למשל, המשתמש יכתוב 2 5 1 3 ונקבל את הקבוצה {1, 2, 5}, או יכתוב 8 7 6 8 4 ונקבל את הקבוצה {6, 7, 8} (ראו דוגמאות נוספות בהמשך.)

uni(on) num1 num2 – מוסיפה לרשימת הפעולות פעולה שהיא איחוד של תוצאות הפעולות שמספרן num1 ו־num2. כאמור, את האיחוד נסמן כ־U. שימו לב בהדפסה לשים סוגריים סביב הפעולות.

inter(section) num1 num2 – מוסיפה לרשימת הפעולות פעולה שהיא החיתוך של תוצאות הפעולות שמספרן num1 ו־num2. כאמור, את החיתוך נסמן כ־^. שימו לב בהדפסה לשים סוגריים סביב הפעולות.

diff(erence) num1 num2 – מוסיפה לרשימת הפעולות פעולה שהיא ההפרש של תוצאות הפעולות שמספרן num1 ו־num2. כאמור, את ההפרש נסמן בסימן המינוס. שימו לב בהדפסה לשים סוגריים סביב הפעולות.

prod(uct) num1 num2 – מוסיפה לרשימת הפעולות פעולה שתיצור קבוצה שהאיברים בה הם תוצאה של הכפלת כל מספר מהתוצאה של הפעולה num1 בכל מספר מהתוצאה של הפעולה num2. למשל, אם התוצאות של שתי הפעולות הן הקבוצות {1, 2} ו־{3, 4}, תוצאת המכפלה היא

{3, 4, 6, 8}. את המכפלה נסמן כ־*, כצפוי. שימו לב בהדפסה לשים סוגריים סביב הפעולות. בנוסף, יש לוודא שהתוצאה נשארת ממויינת ולא כוללת חזרות.

comp(osite) num1 num2 – מוסיפה לרשימת הפעולות פעולה שהיא הרכבה של הפעולות שמספרן num1 ו־num2. לצורך התרגיל נגדיר שתוצאת הפעולה הראשונה תועבר כקלט הראשון לפעולה השנייה. לשם הפשטות, ניתן להדפיס את הפעולה הראשונה והשנייה בזו אחר זו עם חץ מהראשונה לשנייה, כדי לייצג את ההרכבה (ראו בדוגמאות להלן). כמובן שיפה יותר להדפיס את הפעולה הראשונה בתוך הפעולה השנייה (כלומר, במקום הפרמטר הראשון שלה).

del(ete) num – מוחקת את הפעולה שמספרה num מרשימת הפעולות. שימו לב שאם יש פעולות שנסמכות על הפעולה הזו (פעולות שיצרנו תוך שימוש בפונקציה הזו כאבן בניין) הפעולות הללו עדיין צריכות להמשיך לעבוד כרגיל. מצד שני, מספרי הפעולות שנשארו ברשימה צריכים להישאר רציפים, כמו שהוזכר לעיל.

help – מדפיסה מסך עזרה עם רשימת הפקודות האפשריות והסבר קצר עליהן.

exit – מדפיסה למסך "Goodbye" ויוצאת מהתוכנית.

דוגמה לריצת התוכנית:

הערה: אין צורך לעקוב בצורה מדויקת אחרי הניסוח כאן, אבל כן לעקוב אחרי הקו הכללי.

הקלט מהמשתמש מסומן ברקע צהוב. בחלק מהמקרים נוספו הערות (בעברית) להדגשה ותשומת לב.

הפלט ההתחלתי:

```
List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)

Enter command ('help' for the list of available commands): help
The available commands are:
* eval(uate) num ... - compute the result of function #num on the following
set(s); each set is prefixed with the count of numbers to read
* uni(on) num1 num2 - Creates an operation that is the union of operation
#num1 and operation #num2
* inter(section) num1 num2 - Creates an operation that is the intersection
of operation #num1 and operation #num2
* diff(erence) num1 num2 - Creates an operation that is the difference of
operation #num1 and operation #num2
```

```
* prod(uct) num1 num2 - Creates an operation that returns the product of
the items from the results of operation #num1 and operation #num2
* comp(osite) num1 num2 - creates an operation that is the composition of
operation #num1 and operation #num2
* del(ete) num - delete operation #num from the operation list
* help - print this command list
* exit - exit the program

List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)

Enter command ('help' for the list of available commands): eval 0
4 1 2 3 4
3 1 2 5
({ 1, 2, 3, 4 } U { 1, 2, 5 }) = { 1, 2, 3, 4, 5 }

List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)

Enter command ('help' for the list of available commands): uni 0 0

List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)
3.      ((A U B) U (C U D))

Enter command ('help' for the list of available commands): diff 0 1

List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)
3.      ((A U B) U (C U D))
4.      ((A U B) - (C ^ D))

Enter command ('help' for the list of available commands): eval 4
3 1 2 3
5 1 3 5 7 9
3 1 2 3
5 1 3 5 7 9
```

```
(({ 1, 2, 3 } U { 1, 3, 5, 7, 9 }) - ({ 1, 2, 3 } ^ { 1, 3, 5, 7, 9 })) = {
2, 5, 7, 9 }

List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)
3.      ((A U B) U (C U D))
4.      ((A U B) - (C ^ D))

Enter command ('help' for the list of available commands): comp 0 1

List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)
3.      ((A U B) U (C U D))
4.      ((A U B) - (C ^ D))
5.      (A U B) -> (C ^ D)

Enter command ('help' for the list of available commands): eval 5
3 1 2 3
4 1 2 4 5
4 2 3 5 6
({ 1, 2, 3 } U { 1, 2, 4, 5 }) -> ({ 1, 2, 3, 4, 5 } ^ { 2, 3, 5, 6 }) = {
2, 3, 5 }

List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)
3.      ((A U B) U (C U D))
4.      ((A U B) - (C ^ D))
5.      (A U B) -> (C ^ D)

Enter command ('help' for the list of available commands): prod 0 1

List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)
3.      ((A U B) U (C U D))
4.      ((A U B) - (C ^ D))
5.      (A U B) -> (C ^ D)
6.      ((A U B) * (C ^ D))
```

```
Enter command ('help' for the list of available commands): eval 6
2 1 2 2 3 4
3 1 2 3 2 2 3
(({ 1, 2 } U { 3, 4 }) * ({ 1, 2, 3 } ^ { 2, 3 })) = { 2, 3, 4, 6, 8, 9, 12
}

List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)
3.      ((A U B) U (C U D))
4.      ((A U B) - (C ^ D))
5.      (A U B) -> (C ^ D)
6.      ((A U B) * (C ^ D))

Enter command ('help' for the list of available commands): prod 6 0

List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)
3.      ((A U B) U (C U D))
4.      ((A U B) - (C ^ D))
5.      (A U B) -> (C ^ D)
6.      ((A U B) * (C ^ D))
7.      (((A U B) * (C ^ D)) * (E U F))

Enter command ('help' for the list of available commands): del 6
```

(מחקנו את פעולה 6, שעליה התבססה פעולה 7)

```
List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)
3.      ((A U B) U (C U D))
4.      ((A U B) - (C ^ D))
5.      (A U B) -> (C ^ D)
6.      (((A U B) * (C ^ D)) * (E U F))
```

(הפעולות עדיין ממוספרות ברצף)

```
Enter command ('help' for the list of available commands): eval 6
2 1 2 2 3 4
3 1 2 3 2 2 3
1 6
3 8 9 0
```

```
((({ 1, 2 } U { 3, 4 }) * ({ 1, 2, 3 } ^ { 2, 3 })) * ({ 6 } U { 0, 8, 9
})) = { 0, 12, 16, 18, 24, 27, 32, 36, 48, 54, 64, 72, 81, 96, 108 }
```

(ופעולה 6, שהורכבה גם מהפעולה שנמחקה, עדיין עובדת באופן תקין!)

```
List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)
3.      ((A U B) U (C U D))
4.      ((A U B) - (C ^ D))
5.      (A U B) -> (C ^ D)
6.      (((A U B) * (C ^ D)) * (E U F))

Enter command ('help' for the list of available commands): e
Command not found

List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)
3.      ((A U B) U (C U D))
4.      ((A U B) - (C ^ D))
5.      (A U B) -> (C ^ D)
6.      (((A U B) * (C ^ D)) * (E U F))

Enter command ('help' for the list of available commands): eval 9
Operation #9 doesn't exist

List of available set operations:
0.      (A U B)
1.      (A ^ B)
2.      (A - B)
3.      ((A U B) U (C U D))
4.      ((A U B) - (C ^ D))
5.      (A U B) -> (C ^ D)
6.      (((A U B) * (C ^ D)) * (E U F))

Enter command ('help' for the list of available commands): exit
Goodbye!
```

הערות לגבי צורת המימוש:

- אל תחששו להשתמש בכלים של הספרייה הסטנדרטית כמו וקטור ומצביעים חכמים, הם יהפכו את הקוד שלכם להרבה יותר נקי, יעיל וקצר ויקלו עליכם בכתיבת קוד שעובד נכון.
- כדאי להשקיע זמן בתיכון התוכנית. עם תיכון נכון, הקוד יהיה די מינימלי ופשוט לכתיבה.

- האתגר המרכזי מבחינת התיכון הוא לתכנן את מבנה המחלקות והפונקציות שלהן כך שקל יהיה לבצע גם את ההדפסה וגם את החישוב של הרכבת הפעולות בצורה קלה ופשוטה. למשל, ייתכן שיהיה שימושי לייצג דברים נוספים כמחלקות בפני עצמן, לא רק את הפעולות והפקודות המוגדרות כאן ברשימת הדרישות.
- מצד שני, אין צורך להתעמק יותר מידי בירושה בתרגיל הזה, הירושה די פשוטה. העבודה על התיכון היא יותר מבחינת שיתוף הפעולה בין המחלקות ואופן פעולה פשוט.
- ניתן להניח שהקלט תקין מהבחינה שבמקום שאנחנו מצפים למספר אכן מובטח שנקבל מספר ולא אות, שנקבל מספיק קבוצות עבור הפעולה ומספר האיברים בקבוצה אכן יתאים למספר האיברים שהועבר בתחילת הקבוצה, שהמספרים כולם שלמים (אבל לא בהכרח ממוינים או ייחודיים!). לעומת זאת, יש צורך לבדוק את שם הפקודה ואת מספרי הפעולות ואי אפשר להניח שיש כזו פקודה או פעולה. במקרה שהפקודה לא קיימת או שהפעולה לא קיימת יש להדפיס הודעת שגיאה מתאימה ושוב להדפיס את רשימת הפעולות כמו אחרי ביצוע של פקודה תקינה.
- אין חובה שבהדפסה נכתוב שמות שונים לפרמטרים של הפעולות (למשל, חיסור של איחוד וחיתוך אפשר להדפיס כך: ((A U B) - (A ^ B)), למרות שבהפעלת הפעולה יידרשו ארבעה קלטים שונים (שתי ה־A ושתי ה־B שונות זו מזו). כמובן שראוי כן להדפיס שמות שונים, אבל זו לא חובה. הדפסה של שמות שונים באופן עקבי (בדומה לדוגמאות לעיל) תזכה בבונוס של 5 נקודות.
- הקפידו להדפיס את הקבוצה בצורה ממוינת! כדי למיין את מבנה הנתונים שלכם, ניתן (ומומלץ) להיעזר באלגוריתם std::ranges::sort. הוא מקבל כפרמטר את הווקטור למיון וצריך בשבילו #include <algorithm>
- למחיקת כפילויות מהווקטור, כמובן שניתן לבצע זאת ידנית (למשל, העתקה לווקטור חדש תוך דילוג על איברים זהים), אבל ניתן גם להיעזר באלגוריתם: std::ranges::unique. הדרך להשתמש בו על וקטור (ממוין!) בשם v היא:

```
#include <algorithm>
auto [newEnd, end] = std::ranges::unique(v);
v.erase(newEnd, end);
```

(למי שרוצה להבין מה קורה פה: השימוש בו קצת יותר מורכב מהשימוש ב־sort, כי הוא לא מוחק בעצמו את האיברים המיותרים מהווקטור, רק "מזיז" להתחלה את מה שצריך להישאר ומחזיר מידע שאומר מאיפה עד איפה נשארו איברים מיותרים שצריך למחוק).

- את הפעולות השונות על הקבוצות, איחוד, חיתוך והפרש, אפשר לכתוב "ידנית", אבל ניתן (ומומלץ) להיעזר באלגוריתמים מהספרייה הסטנדרטית במקום לכתוב זאת בעצמנו. האלגוריתמים הרלוונטיים הם: std::ranges::set_union, std::ranges::set_intersection ו־std::ranges::set_difference. שימו לב שהפרמטרים שהם מקבלים הם שני מבני נתונים (למשל, std::vector) שחייבים להיות ממוינים! כמו כן, האלגוריתמים האלה מסתמכים על

אופרטורים סטנדרטיים כמו ==, כדי לקבוע אם איברים זהים זה לזה. דוגמה לאיחוד של הווקטורים v1 ו־v2 (ששניהם מחזיקים int־ים):

```
#include <algorithm>
#include <iterator>
auto result = std::vector<int>();
std::ranges::set_union(v1, v2, std::back_inserter(result));
```

- אם נרצה להפעיל את האלגוריתמים על מחלקה שלנו שמחזיקה וקטור (למשל, מחלקה שמייצגת קבוצה), הדרך הכי פשוטה היא להוסיף למחלקה את השורות הבאות (בהנחה ששם הווקטור המוחזק במחלקה הוא m_v):

```
auto begin() const { return m_v.begin(); }
auto end() const { return m_v.end(); }
```

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 שיצרנו, 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. נכתוב 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. 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.

<u>בהצלחה!</u>