234124 - מבוא לתכנות מערכות
תרגיל בית מספר 1
סמסטר אביב 2023

תאריך פרסום: 16.4.23
תאריך הגשה: 7.5.23 בשעה 23:59
מתרגל אחראי לתרגיל: אביתר זיו

1 הערות כלליות

- **תרגיל זה מהווה 6% מהציון הסופי**
- התרגיל להגשה **בזוגות** בלבד.
- מענה לשאלות בנוגע לתרגיל יינתן אך ורק בפורום התרגיל בפיאצה או בסדנות. לפני פרסום שאלה בפורום אנא בדקו אם כבר נענתה – מומלץ להיעזר בכלי החיפוש שהוצגו במצגת האדמיניסטרציה בתרגול הראשון.
- שימו לב: לא תינתנה דחיות במועד הגשת התרגיל פרט למקרים חריגים. **תכננו את הזמן בהתאם**.
- ראו את התרגיל עד סופו לפני שאתן מתחילות לממש. חובה להתעדכן בעמוד ה-F.A.Q של התרגיל, הכתוב שם מחייב.
- העתקות קוד בין סטודנטים ובפרט גם העתקות מסמסטרים קודמים תטופלנה. עם זאת – מומלץ ומבורך להתייעץ עם חברים על ארכיטקטורת המימוש.
- קבצי התרגיל נמצאים בGitHub repository הבא: https://github.com/CS234124/ex1

2 חלק יבש – זיהוי שגיאות בקוד

2.1 *סעיף א'*

מצאו 6 שגיאות תכנות ו-2 שגיאות קונבנציה[1] (code convention) בפונקציה הבאה. מטרת הפונקציה היא לשכפל מספר פעמים את המחרוזת המתקבלת לתוך מחרוזת חדשה. למשל, הקריאה stringDuplicator("Hello",3) תחזיר את המחרוזת "HelloHelloHello". במקרה של שגיאה בריצת הפונקציה, הפונקציה תחזיר NULL. מותר להניח שהקלט תקין, כלומר אין צורך בבדיקה תקינות קלט כפי שלמדתם בתרגולים.

```
#include "stdlib.h"
#include "string.h"
#include "assert.h"

char* stringDuplicator(char* s, int times){
      assert(!s);
      assert(times > 0);
      int LEN = strlen(s);
      char* out = malloc(LEN*times);
      assert(out);
      for (int i=0; i<=times; i++){
      out = out + LEN;
      strcpy(out,s);
      }
      return out;
}
```

2.2 *סעיף ב'*

כתבו גרסה מתוקנת של הפונקציה.

[1] ראו מסמך "Code Conventions.pdf" באתר הקורס

3 חלק רטוב

לאחר הפריצה למחשבי הטכניון, ההאקרים של דארקביט מעוניינים לנצל את השליטה שהשיגו במערכת על מנת להשתלט על תורי הרישום למקצועות. בחלק זה נממש תור ישראלי, ובעזרתו נעזור להאקרים לממש כלי להשתלטות על תורי הרישום.

תור ישראלי הוא מבנה נתונים, שכמו תור רגיל מסדר איברים בצורת FIFO. כלומר, ניתן בכל עת להכניס איברים לתור, ובכל זמן שהתור לא ריק ניתן להוציא ממנו את האיבר הראשון שהוכנס אליו, מבין אלו שעדיין בתוכו. תור ישראלי הוא תור, שבו איבר שמוכנס לתור יכול להיות מוגדר כמי שנכנס לפני איברים שכבר היו בתור (ובכך "לעקוף" אותם ולצאת לפניהם) אם יש איבר אחר בתור שהוא "חבר" של האיבר הנכנס, ובמקרה זה האיבר הנכנס נחשב כאיבר שנכנס מיד אחרי אותו חבר. אנחנו נממש תור ישראלי, עם מספר שינויים. בניגוד לתור הישראלי הקלאסי, אנחנו נאפשר לכל איבר מכסה קבועה של 5 חברים, ומעליה האיבר לא יוכל להכניס חברים נוספים. בנוסף, נגדיר שני איברים אויבים, כך שאיבר בתור יכול למנוע ממכסה קבועה 3 של אויבים לעקוף אותו בתור, גם אם יש להם חברים לפניו. במקרה של מספר יריבים המונעים מאיבר להיכנס למיקום, היריב הראשון בתור, מבחינת קרבה לראש התור, הוא המונע, ולו נספרת המניעה במניין היריבים שחסם.

3.1 רקע

3.1.1 חברות

בהינתן מערך של מצביעים לפונקציות מטיפוס int (void*,void*) (שתיקראנה *פונקציות חברות*), נגדיר שכל אחת מהן, בהינתן שני אובייקטים, מחזירה את "מידת החברות" של שניהם (מספר שלם עליו לא ניתן להניח דבר).
שני אובייקטים יקראו *חברים* אם קיימת במערך פונקציה שעל פיה מידת החברות שלהם גדולה מפרמטר מסויים (*סף חברות*).
בנוסף, שני אובייקטים ייקראו *אויבים* אם הם לא חברים לפי כל הפונקציות במבנה, וממוצע "מידות החברות" שלהם קטן מפרמטר אחר (*סף יריבות*).

3.1.2 ממשק התור

במקרה של סתירה בין תיאור הממשק פה וקובץ ה־header שסופק לכם, קובץ ה־header הוא הקובע

1. **IsraeliQueue IsraeliQueueCreate(FriendshipFunction *, ComparisonFunction, int friendship_th, int rivalry_th)**

   מחזיר תור ישראלי חדש, עם פונקציות החברות ופונקציית ההשוואה שסופקה, וספי החברות והאיבה שניתנו כפרמטרים. מובטח שמערך פונקציות החברות יסתיים ב־NULL.

2. **IsraeliQueueError IsraeliQueueEnqueue(IsraeliQueue q, void* item)**

   מכניסה את האיבר המוצבע ע"י item למקום הראשון בתור אליו הוא יכול להיכנס.

3. **IsraeliQueueError IsraeliQueueAddFriendshipMeasure(IsraeliQueue q, FriendshipFunction friendships_function)**

   מוסיפה לתור "מידת חברות" חדשה

4. **IsraeliQueueError IsraeliQueueUpdateFriendshipThreshold(IsraeliQueue q, int n_thresh)**

   מעדכנת את ערך הסף לחברות

5. **IsraeliQueueError IsraeliQueueUpdateRivalryThreshold(IsraeliQueue q, int n_thresh)**

   מעדכנת את ערך הסף לאיבה

6. **int IsraeliQueueSize(IsraeliQueue q)**

   מחזירה את גודל התור (מספר האיברים בו)

7. **void* IsraeliQueueDequeue(IsraeliQueue q)**

   מחזירה את האיבר בראש התור ומסירה אותו ממנו

8. **int IsraeliQueueContains(IsraeliQueue q, void* item)**

מחזירה 1 אם קיים בתור איבר ששווה (לפי פונק ההשוואה) ל־item, אחרת 0

9. **IsraeliQueueError IsraeliQueueImprovePositions(IsraeliQueue q)**

כל איברי התור מהסוף להתחלה מתקדמים למקום הקדמי ביותר בתור אליו הם יכולים להגיע (חישבו מתי פעולה זו אינה טריוויאלית)

שימו לב: מיקומו הנוכחי של איבר נחשב כמקום אליו הוא יכול להגיע, אלא אם כן ישנו אויב בתור שחוסם אותו

הבהרה: מספר החסימות והקידומים שביצע איבר בתור **לא** יתאפס כתוצאה מהפעלת הפונקציה.

10. **IsraeliQueue IsraeliQueueMerge(IsraeliQueue *qarr, ComparisonFunction compare_function)**

ממזג מערך של תורים ישראליים לתור חדש, כך שאובייקטים מכל תור מוכנסים בסבב. לתור החדש תהיינה את כל "מידות החברות" של התורים הממוזגים, סף החברות בו יהיה ממוצע ספי החברות בתורים הממוזגים, וסף האיבה בו יהיה ערך שלם עליון של הממוצע ההנדסי של הערכים המוחלטים של ספי האיבה של התורים. פונקציית ההשוואה של התור החדש מתקבלת כפרמטר לפונקציה. המערך qarr יסתיים ב־NULL.

3.2 *מימוש תור ישראלי*

ממשו את התור הישראלי שתואר בממשק.
הממשק שתואר בסעיף 3.1 נתון לכם בקובץ הממשק IsraeliQueue.h. עליכם לממש את הממשק המתואר בקובץ IsraeliQueue.c.

כמו כן, מסופק לכם טסט בסיסי בקובץ IsraeliQueue_example_test.c, הנועד ל"בדיקת שפיות" בסיסית ומשמש דוגמה עבור כתיבת הטסטים שלכם.

דגשים נוספים ודרישות מימוש

- **קראו את התיעוד ב-IsraeliQueue.h! הוא מגדיר במפורש כל פעולה שעליכם לממש ויעזור לכם במיוחד!**
- קיימות פונקציות שלהן מספר ערכי שגיאה אפשריים. בהערה מעל כל פונקציה בקובץ IsraeliQueue.h תוכלו למצוא את כל השגיאות שיכולות להתרחש בעת קריאה אליה בקובץ הממשק שמסופק לכם. **במקרה של כמה שגיאות אפשריות החזירו את השגיאה שהוגדרה ראשונה בקובץ.**
  אם מתרחשת שגיאה שאינה ברשימה, יש להחזיר ISRAELI_QUEUE_ERROR.
- אין הגבלה על מספר האיברים בתור.
- במקרה של שגיאה יש לשמור על שלמות מבנה הנתונים ולוודא שאין דליפות זיכרון.

3.3 *מימוש מערכת לניהול תורי רישום לקורסים מומלצים*

חברי דארקביט הצליחו אמנם לפרוץ למערכות הטכניון, אולם העבודה הכרוכה במתקפת סייבר במהלך תקופת המבחנים פגעה בהישגים שלהם במבחנים. על מנת "לרפד" את הממוצע, החליטו ההאקרים להירשם לקורסים קלים בפקולטות לא נחשבות.

לאור הביקוש הרב לקורסים אלו, החליטו ההאקרים להחליף את תורי הרישום למקצועות בתורים ישראליים, וכך לעקוף את שאר הסטודנטים.

בחלק זה יש לכתוב את קובץ הממשק HackerEnrollment.h עם הפונקציות הבאות, ולממשן בקובץ HackerEnrollment.c:

EnrollmentSystem createEnrollment(FILE* students, FILE* courses, FILE* hackers)

תיצור מבנה EnrollmentSystem שיכיל את הסטודנטים והקורסים בקבצים. מבנה קובץ הסטודנטים:

<Student ID> <Total Credits> <GPA> <Name> <Surname> <City> <Department>\n

כאשר GPA ערך בין 0 ל־100, Total Credits מספר שלם אי שלילי ו־Student ID מחרוזת בת תשע ספרות. מבנה קובץ ההאקרים:

<Student ID> \n
<Course Numbers>*\n //Desired courses
<Student ID>*\n //Friends
<Student ID>*\n //Rivals

לצורך שמירה על דיסקרטיות, החליטו שכל האקר יצטרף לסטודנט לפי מידות החברות הבאות:

- ההאקר והסטודנט חברים לפי קובץ ההאקרים (מידה טרנארית – 20 חברים, 20- אויבים, 0 לא זה ולא זה)
- "מרחק השם" – סכום הפרשי ערכי ה־ASCII של שמותיהם
- הפרש מספרי הזהות

מבנה קובץ הקורסים מוגדר כדלקמן:

<Course Number> <Size>\n

EnrollmentSystem readEnrollment(EnrollmentSystem sys, FILE* queues)

הפונקציה קוראת קובץ המתאר את תורי הרישום למקצועות. פורמט הקובץ:

<Course Number> <Student IDs>*

ותחזיר מבנה EnrollmentSystem המייצג את תורי הרישום

void hackEnrollment(EnrollmentSystem sys, FILE* out)

תכתוב ל־out תורי רישום חדשים, בהם כל האקר נמצא במקום <Size בשני קורסים שביקש, או את ההודעה:

Cannot satisfy constraints for <Student ID>+

כאשר מספרי הזהות יזהו האקרים שלא קיבלו שני קורסים שביקשו.

3.4 תוכנית HackEnrollment

בחלק זה נממש תוכנית בשם HackEnrollment שמאפשרת למשתמש לשנות תורי רישום. המימוש של פונקציית ה־main יהיה בקובץ main.c.

התוכנית תקבל דרך ה־command line interface חמישה ארגומנטים:

flag – אופציונאלי, מגדיר האם להתייחס לאותיות גדולות בשמות הסטודנטים בחישוב מרחק בין השמות, כאשר דיפולטיבית מתייחסים לגודל:

i- – לא להתייחס

students – קובץ הסטודנטים.

courses – קובץ הקורסים

hackers – קובץ ההאקרים

target – קובץ פלט שאליו רוצים לכתוב את התוצאה

פורמט הפקודה:

- ./HackEnrollment <flags> <students> <courses> <hackers> <queues> <target>

3.5 *דגשים ודרישות מימוש*

- המימוש חייב לציית לכללי כתיבת הקוד המופיעים תחת Course Material -> Code Conventions.
אי עמידה בכללים אלו תגרור הורדת נקודות.
- על המימוש שלכם להתבצע ללא שגיאות זיכרון (גישות לא חוקיות וכדומה) וללא דליפות זיכרון.

אם בפונקציה מסוימת קיימות מספר אפשרויות לערך שגיאה, אנו נבחר את השגיאה הראשונה על פי סדר השגיאות המופיע תחת הפונקציה הספציפית. השגיאה OUT_OF_MEMORY יכולה להתרחש בעת כישלון בהקצאת זיכרון בכל שלב ולא לא משתתפת בעניין הסדר.

- **המערכת צריכה לעבוד על שרת CSL3.**

3.6 *Makefile*

עליכם לספק Makefile כפי שנלמד בקורס עבור בניית הקוד של תרגיל זה.

- הכלל הראשון ב Makefile יקרא program ויבנה את התוכנית HackEnrollment המתוארת למעלה. יש לכתוב את הקובץ כפי שנלמד וללא שכפולי קוד.
- אנו מצפים לראות שלכל מודול קיים כלל אשר בונה עבורו קובץ o., דבר שכפי שלמדתם בקורס - אמור לחסוך הידור של כל התכנית כאשר משנים רק חלק קטן ממנו.
- הוסיפו גם כלל clean.
- תוכלו לבדוק את ה-Makefile שלכם באמצעות הרצת הפקודה make והפעלת קובץ ההרצה שנוצר בסופו.

3.7 *הידור קישור ובדיקה*

התרגיל ייבדק על שרת csl3 ועליו לעבור הידור בעזרת הפקודה הבאה:

```
gcc -std=c99 -o HackEnrollment -I/home/mtm/public/2223b/ex1 -Itool -Wall -pedantic-errors -Werror -DNDEBUG *.c tool/*.c
```

פירוט תפקיד כל דגל בפקודת ההידור (לא חייבים להבין עד הסוף):

- std=c99 - שימוש בתקן השפה c99.
- [program name] o- – הגדרת שם הקובץ המהודר.
- Wall- – דווח על כל האזהרות.
- pedantic-errors- – דווח על סגנון קוד שאינו עומד בתקן הנבחן כשגיאות.
- Werror- – התייחס לאזהרות כאל שגיאות – משמעות דגל זה שהקוד חייב לעבוד הידור ללא אזהרות ושגיאות.
- DNDEBUG- -מוסיף את השורה #define NDEBUG בתחילת כל יחידת קומפילציה. בפועל מתג זה יגרום לכך שהמאקרו assert לא יופעל בריצת התוכנית.
- *.c קבצי הפתרון שלכם.
- tool/*.c קבצי הפתרון שלכם שנמצאים בתיקיית tool וממשים את HackEnrollment.
- Itool– המטרה של חלק זה היא לכלול את התקייה tool במסלול החיפוש של קבצי ההידור.
- I/home/mtm/public/2223b/ex1 – לכלול את התיקייה במסלול החיפוש של קבצי ההידור. זוהי התיקיית שמכילה את הקבצים המסופקים לכם.

לנוחיותכם, מסופקת לכם תוכנית "בדיקה עצמית" בשם finalCheck, התוכנית בודקת שקובץ ההגשה, קובץ ה-zip בנוי נכון ומריצה את הטסטים סופקו כפי שירוצו על ידי הבודק האוטומטי. הפעלת התוכנית ע"י הפקודה:

```
~mtm/public/2223b/ex1/finalCheck <submission>.zip
```

הקפידו להריץ את הבדיקה על קובץ ההגשה. אם אתה משנים אותו, הקפידו להריץ את הבדיקה שוב.

3.8 *ולגרינד ודליפות זיכרון*

המערכת חייבת לשחרר את כל הזיכרון שעמד לרשותה בעת ריצתה. על כן עליכם להשתמש ב-valgrind שמתחקה אחר ריצת התכנית שלכם, ובודק האם ישנם משאבים שלא שוחררו. הדרך לבדוק האם יש לכם דליפות בתכנית היא באמצעות שתי הפעולות הבאות (שימו לב שחייב להיות main, כי מדובר בהרצה ספציפית):

1. קימפול של השורה לעיל עם הדגל g-.
2. הרצת השורה הבאה:

   valgrind --leak-check=full ./HackEnrollment

   כאשר HackEnrollment הוא שם קובץ ההרצה.

הפלט ש-valgrind מפיק אמור לתת לכם, במידה ויש לכם דליפות, את שרשרת הקריאות שהתבצעו וגרמו לדליפה. אתם אמורים באמצעות דיבוג להבין היכן היה צריך לשחרר את אותו משאב שהוקצה ולתקן את התכנית. בנוסף, valgrind מראה דברים נוספים כמו פנייה לא חוקית לזיכרון (שלא בוצע בעקבותיה segmentation fault) – גם שגיאות אלו עליכם להבין מהיכן הן מגיעות ולתקן.

3.9 *בדיקת התרגיל*

התרגיל ייבדק בדיקה יבשה (מעבר על קונבנציות הקוד והארכיטקטורה) ובדיקה רטובה.
הבדיקה היבשה כוללת מעבר על הקוד ובודקת את איכות הקוד (שכפולי קוד, קוד מבולגן, קוד לא ברור, שימוש בטכניקות תכנות "רעות").
הבדיקה הרטובה כוללת את הידור התכנית המוגשת והרצתה במגוון בדיקות אוטומטיות. על מנת להצליח בבדיקה שכזו, על התוכנית לעבור הידור, לסיים את ריצתה, ולתת את התוצאות הצפויות ללא דליפות זיכרון.

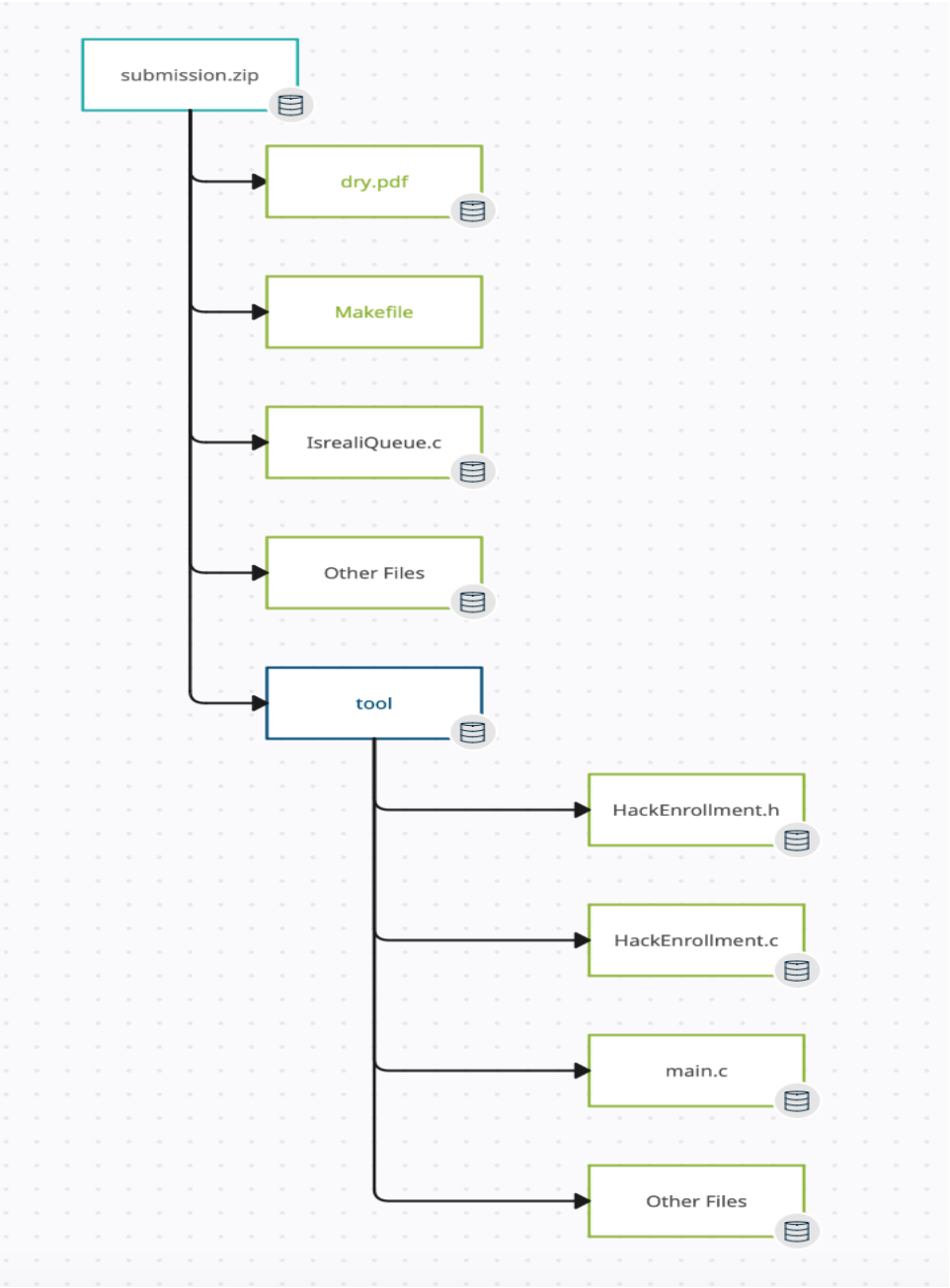
4 הגשה

את ההגשה יש לבצע דרך אתר הקורס, תחת

Assignments -> HW1 -> Electronic Submit.

הקפידו על הדברים הבאים:

- יש להגיש את קבצי הקוד וה-makefile מכווצים לקובץ zip (לא פורמט אחר) כאשר כל הקבצים עבור פתרון **המערכת לדחיסת תמונות** מופיעים בתיקיית tools בתוך קובץ ה zip. הקבצים עבור **החלק הראשון** יופיעו בתוך תיקיית השורש.
- יש להגיש קובץ PDF עבור החלק היבש, קראו לקובץ זה בשם dry.pdf ולשים אותו בתיקיית השורש בתוך קובץ ה zip (ליד ה-makefile).
- **אין להגיש אף קובץ מלבד קבצי h וקבצי c אשר כתבתם בעצמכם ואת ה-makefile אשר נדרשתם לעשות ואת ה-PDF של היבש.**
- הקבצים אשר מסופקים לכם יצורפו על ידנו במהלך הבדיקה, וניתן להניח כי הם יימצאו בתיקייה הראשית.
- ניתן להגיש את התרגיל מספר פעמים, רק ההגשה האחרונה נחשבת.
- על מנת לבטח את עצמכם נגד תקלות בהגשה האוטומטית שימרו את קוד האישור עבור ההגשה. עדיף לשלוח גם לשותף.
  - כל אמצעי אחר לא יחשב הוכחה לקיום הקוד לפני ההגשה.
  - מצורף תרשים של מבנה קובץ ההגשה:


submission.zip
dry.pdf
Makefile
IsrealiQueue.c
Other Files
tool
HackEnrollment.h
HackEnrollment.c
main.c
Other Files

בהצלחה !