ה אוניברסיטה העברית בירושלים

בית הספר להנדסה ולמדעי המחשב ע"ש רחל וסלים בנין

סדנת תכנות C ו- C++ - תרגיל 1

<u>נושאי התרגיל:</u> היכרות עם השפה, קומפילציה, משתנים, אריתמטיקה פשוטה, פלט, תנאים, לולאות, פונקציות ושימוש ב- CLI, Test Driven Development(TDD)

תאריך הגשה: יום רביעי ה-29.5 בשעה 22:00

1 רקע

קריפטוגרפיה הינה תחום עתיק, עבורו ניתן למצוא תיעוד משחר ההיסטוריה. בעבר, נעשה שימוש בקריפטוגרפיה בעיקר על ידי הצבא והמלוכה, בעוד כיום כל אחד מאיתנו עושה בה שימוש על בסיס יום יומי, למשל בעת שימוש במחשב האישי שלנו (או במכשיר החכם הנייד) – וזאת אף מבלי לשים לב לכך.

בתרגיל זה נממש תוכנה cipher המצפינה ומקודדת טקסט באמצעות צופן הנקרא "צופן קיסר" על שמו של יוליוס קיסר, או "צופן היסט ".

פיתוח מבוסס-בדיקות: בתרגיל זה נמליץ לעבוד בהליך פיתוח מבוסס-בדיקות (TDD). בהליך זה, אנו בוחנים את דרישות התוכנית ובונים בדיקות לתכנית (tests) **לפני** שעובדים על פיתוח התוכנית המרכזית. כך מבטיחים שהתכנית עומדת במפרט הדרישות. הסבר על שיטה זו, בהמשך.

הערה ממני אליכם: בסוף קובץ התרגיל ישנם מספר נספחים שיכולים להקל עליכם בזמן העבודה עם clion, ממליץ בחום לקרוא אותם בהתחלה.

2 תהליך העבודה המומלץ

בתרגיל זה אנו מספקים חמישה קבצי שלד המרכיבים את התוכנה, מומלץ מאוד להוריד אותם ולא להעתיק ידנית (אחרת עלול ליצור בעיות עם תווים בלתי נראים). הקבצים הם:

- **cipher.h**
- **cipher.c**
- **main.c**
- **tests.h**
- **tests.c**

אנחנו ממליצים להתחיל ראשית מלעבוד על הטסטים לcipher. אז לממש את cipher, תוך כדי בדיקה עם הטסטים שהכנתם מראש ואז להתקדם לבדיקת הקלט.

נסביר כעת איך להכין את הטסטים, לאחר מכן נעבור על איך cipher.c עובד ולבסוף נעבור על בדיקת תקינות קלט.

מומלץ לבצע שמירה תכופה של השינויים תוך כדי העבודה על התרגיל באמצעות git commit. קריאה נוספת: https://git-scm.com/doc

3 Test Driven Development (TDD)

את המטלה הזאת אנחנו נמליץ (אבל לא נחייב) לממש בהליך פיתוח מבוסס בדיקות-תוכנה TDD. שיטת עבודה נפוצה מאוד שבה אחרי קבלת הדרישות של המטלה אנחנו קודם רושמים את הבדיקות שאיתן נבדוק את הקוד, ורק אח"כ כותבים את הקוד עצמו.

וביותר פירוט – בשיטת העבודה הזאת ישנם 5 שלבים עיקריים עליהם נחזור עד להשלמת מפרט הדרישות:

- **הוספת בדיקה:** כותבים בדיקה (טסט) לפי מפרט הדרישות עבור אלמנט בודד אותו נרצה להוסיף לתכנית. הבדיקה תיכשל אם ורק אם אותו אלמנט אינו תקין. אם בתכנית ישנם מספר חלקים, לרוב מומלץ לכתוב את הבדיקות ואת הקוד הרלוונטי לכל חלק בנפרד.
- **הרצת כל הבדיקות:** הבדיקה האחרונה שהתווספה חייבת להיכשל, ואך ורק היא. אם צריך, משפרים את הבדיקה.
- **כתיבת קוד:** כותבים את הקוד הפשוט ביותר אשר עובר את כל הבדיקות שנכתבו עד כה.
- **הרצת כל הבדיקות:** כל הבדיקות אמורות לעבור בהצלחה. אם צריך, משפרים את הקוד.
- **שיפור הקוד:** משפרים ומשפצים את הקוד כך שיהיה קריא וקל לתחזוקה. בעזרת הבדיקות, מוודאים ששום דבר לא נהרס.

את השלבים האלו מבצעים עד שעומדים בכל מפרט הדרישות.

לקריאה נוספת: https://en.wikipedia.org/wiki/Test-driven_development

כעת נבין מה הבעיה שצריך לפתור == הקוד שצריך להכין!

4 צופן קיסר (צופן היסט)

נתחיל עם כמה הגדרות שישמשו אותנו לאורך מסמך זה:

- ערך הזחה - מספר $k \in Z$ אשר יהיה המפתח של ההצפנה והפענוח (נסביר איך הוא עובד עוד מעט).
- עבור מחרוזת s באורך n, נסמן $s = c_0 c_1 \cdots c_{n-1}$.

הצופן מורכב מ-3 רכיבים:

מפתח הצפנה/פענוח: מספר שלם k.

מצפין: מערכת אשר יודעת לקבל מפתח k ומחרוזת s ומצפינה אותה בעזרת k. – בשבילנו פונקציית cipher

מפענח: מערכת אשר יודעת לקבל מפתח הצפנה k ומחרוזת מוצפנת s ומפענחת אותה. – בשבילנו פעולת decipher

הקוד אותו נרצה לערוך נמצא בקובץ cipher.c, **אין סיבה לערוך את** cipher.h.

אז איך מצפינים?

דרך הפעולה של צופן קיסר

המצפין, מקבל מחרוזת, s, ערך הזחה k. עבור כל תו $c_i \in s$, **אם** c_i הוא אות באלפבית שלנו, אזי "נזיח ימינה" את c_i, k פעמים. למשל, אם $c_i = 'A'$ ובנוסף $k = 2$, אז נזיח את c_i פעמיים, פעם ראשונה ל-'B' ושנייה ל-'C'.

נציין כי בתוכנה שלנו נעבוד רק עם האלפבית האנגלי, פירוט בהמשך, וכעת עוד דוגמאות:

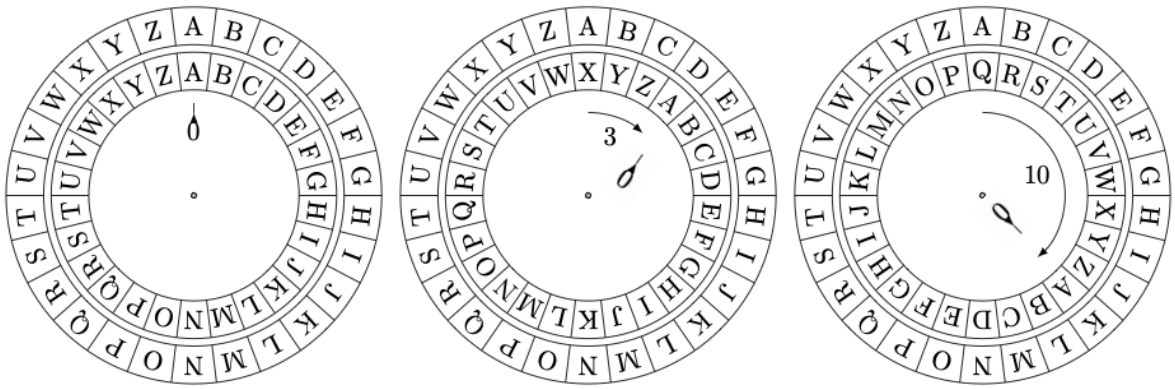
כאן למשל זו דוגמא ציקלית (שעשינו סיבוב שלם) - cipher(“Z”, 3) = “C” cipher(“E”, 3) = “H”

סימן קריאה הוא לא חלק מהאפלבית ולכן הוא נשאר - cipher(“AB!”, 2) = “CD!”

המפענח עבורנו הינה פעולת decipher, אשר מקבלת בדיוק את אותם הערכים כמו המצפין. ההבדל המרכזי היא שמזיחה לכיוון ההפוך מפעולת cipher. למשל, אם $c_i = 'A'$ ובנוסף $k = 2$, אז נזיח את c_i פעמיים **שמאלה**, פעם ראשונה ל-'Z' ושנייה ל-'Y'.

decipher(“A”, 3) = “X” decipher(“D”, 3) = “A”

לסיום, בתקווה שהדבר יפשט את הדברים, שימו לב לאילוסטרציה הבאה:



סדר האלפבית אחרי הזחה של 10 סדר האלפבית אחרי הזחה של 3 סדר האלפבית המקורי

נספח נמצא פה, בשביל הסבר מפורט יותר ופורמלי.

ASCII 5

כפי שלמדנו בכיתה, בשפת C כל תו באלפבית האנגלי (כולל סימני פיסוק וכן הלאה) מיוצג באמצעות מספר. שיטת הקידוד הבסיסית נקראת ASCII, ובה כל תו מתאים למספר בין 0 ל-127. הקידוד הזה מאפשר לנו להשוות בין אותיות במחשב בקלות. הינה דוגמה שתוכל להקל עליכם את כתיבת התוכנה:

```
printf("%d", 'A');
```

המדפיסה: 65

טבלת ASCII נמצאת בקישור הבא לנוחותכם: https://www.asciitable.com

6 התוכנה cipher

התוכנה שנממש, cipher, מאפשרת להצפין ולפענח קטעי טקסט באמצעות צופן קיסר. התוכנה תהיה מורכבת מחמישה קבצים, קובץ שלד לכל אחד מהם נמצא במודל:

- **cipher.h**
- **cipher.c**
- **main.c**
- **tests.h**
- **tests.c**

וכעת לעבודה! קודם כל רבותיי, טסטים!

6.1 tests.h (קיים שלד לנוחיותכם במודל)

בקובץ זה יש את הצהרות הפונקציות שתצטרכו לממש בקובץ המימושים tests.c. **אין צורך לעשות שינויים בקובץ זה.**

6.2 tests.c (קיים שלד לנוחיותכם במודל)

בקובץ זה תצטרכו לממש את הפונקציות המוצהרות בקובץ h.

בכל הפונקציות: הפונקציה תחזיר 0 אם הפונקציה הנבדקת עשתה את הנדרש (כלומר קיבלתם את התוצאה הרצויה של המחרוזת שלכם אחרי שהפעלתם את הפונקציה) או תחזיר 1 אם הפונקציה לא החזירה את הנדרש.
~~למה 0 עבור נכון? כי זה שווה לערך true בספריית stdbool שכדאי לעבוד איתה.~~ **<--טעות, לא להתייחס**

בנוסף, בכל הפונקציות אתם אמורים לחשוב על הקלט של הפונקציות הנבדקות וגם על הפלט שלהן (ראו דוגמא בקובץ השלד שקיבלתם)

6.2.1 כתיבת טסטים עבור cipher:

```
int test_cipher_non_cyclic_lower_case_positive_k()
```

הפונקציה תבצע בדיקה של הפונקציה cipher עבור מחרוזת **שאינה** משתמשת במעגליות בהצפנה עם k=3. (למשל המחרוזת bax משתמשת במעגליות עם k=3 מכיוון ש-x יהפוך ל a, אבל המחרוזת wat אינה צריכה צקליות עם k=3) . השתמשו **באותיות קטנות בלבד. ניתן לראות פתרון לדוגמה בשלד.**

```
int test_cipher_cyclic_lower_case_special_char_positive_k()
```

הפונקציה תבצע בדיקה של הפונקציה cipher עבור מחרוזת אשר כן משתמשת במעגליות בהצפנה עם k=2

בפונקציה זו אתם נדרשים להשתמש במילה **שמכילה סימנים** אשר אמורים להישאר אחרי הצפנה. למשל נקודה, פסיק או רווח.

```
int test_cipher_non_cyclic_lower_case_special_char_negative_k()
```

הפונקציה תבצע בדיקה של הפונקציה cipher עבור מחרוזת **שאינה** משתמשת במעגליות בהצפנה עם k=-1.

בפונקציה זו אתם נדרשים להשתמש במילה **שמכילה סימנים.**

```
int test_cipher_cyclic_lower_case_negative_k()
```

הפונקציה תבצע בדיקה של הפונקציה cipher עבור מחרוזת אשר כן משתמשת במעגליות בהצפנה עם k=-3.

בפונקציה זו אתם נדרשים להשתמש **באותיות קטנות בלבד.**

```
int test_cipher_cyclic_upper_case_positive_k()
```

הפונקציה תבצע בדיקה של הפונקציה cipher עבור מחרוזת אשר כן משתמשת במעגליות בהצפנה עם k=29.

בפונקציה זו אתם נדרשים להשתמש **באותיות גדולות בלבד.**

6.2.2 כתיבת טסטים עבור decipher:

```
int test_decipher_non_cyclic_lower_case_positive_k()
```

הפונקציה תבצע בדיקה של הפונקציה decipher עבור מחרוזת **שאינה** משתמשת במעגליות בפענוח עם k=3.

השתמשו **באותיות קטנות בלבד** בפונקציה זו. **ניתן לראות פתרון לדוגמה בשלד.**

```
int test_decipher_cyclic_lower_case_special_char_positive_k()
```

הפונקציה תבצע בדיקה של הפונקציה decipher עבור מחרוזת אשר כן משתמשת במעגליות בפענוח עם k=2

בפונקציה זו אתם נדרשים להשתמש במילה **שמכילה סימנים.**

```
inttest_decipher_non_cyclic_lower_case_special_char_negative_k()
```

הפונקציה תבצע בדיקה של הפונקציה decipher עבור מחרוזת **שאינה** משתמשת במעגליות בפענוח עם k=-1.

בפונקציה זו אתם נדרשים להשתמש במילה **המכילה סימנים.**

```
int test_decipher_cyclic_lower_case_negative_k()
```

הפונקציה תבצע בדיקה של הפונקציה decipher עבור מחרוזת אשר כן משתמשת במעגליות בפענוח עם k=-3.

בפונקציה זו אתם נדרשים להשתמש **באותיות קטנות בלבד.**

```
int test_decipher_cyclic_upper_case_positive_k()
```

הפונקציה תבצע בדיקה של הפונקציה decipher עבור מחרוזת אשר כן משתמשת במעגליות בפענוח עם k=29.

בפונקציה זו אתם נדרשים להשתמש **באותיות גדולות בלבד.**

הערה: בכל קובץ מסוג c. , בשביל להשתמש בפונקציות הממומשות בקובץ אחר, נצטרך לייבא את קבצי ה-h המכילים את הצהרות הפונקציות הללו. קבצי השלד הנמצאים במודל, כבר מכילים את שורות היבוא הנדרשות.

שימו לב: הבדיקות אשר ביקשנו ממכם לא בהכרח יספיקו על מנת לעמוד בכל דרישות התרגיל. אם תרצו הוסיפו בדיקות נוספות כרצונכם על מנת לוודא שהתרגיל שלכם עובד כראוי (אך אל תגישו בדיקות אלו).

כעת, לאחר שהכנו את כלל הטסטים שלנו, נעבור לקבצי ה-cipher!

6.3 cipher.h (קיים שלד לנוחיותכם במודל)

בקובץ זה נמצאות הצהרות לפונקציות שאותן תצטרכו לממש בקובץ המימושים cipher.c.

אין צורך לעשות שינויים בקובץ זה.

6.4 cipher.c (קיים שלד לנוחיותכם במודל)

בקובץ זה תצטרכו לממש את הפונקציות הבאות:

```
void cipher(char s[], int k)
```

הפונקציה תקבל שני פרמטרים:

- s- הינו המחרוזת שאנחנו רוצים להצפין.
- k- פרמטר ההזחה כפי שהוגדר קודם.

הפונקציה **אינה מחזירה כלום, ומשנה** את s להכיל את ההצפנה של הקלט.

```
void decipher(char s[], int k)
```

הפונקציה תקבל שני פרמטרים:

- s- הינו המחרוזת שאנחנו רוצים לפענח.
- k- פרמטר ההזחה כפי שהוגדר קודם.

הפונקציה **אינה מחזירה כלום, ומשנה** את s להכיל את הפענוח של הקלט.

ולבסוף, נחבר הכל יחדיו ונוודא קלט ופלט!

6.5 main.c(קיים שלד לנוחיותכם במודל)

קובץ זה יהיה הקובץ הראשי של התוכנית, להלן נציין את אוסף הדרישות מהתכנית שלכם. אתם רשאים (אך לא חייבים) להוסיף פונקציות עזר כרצונכם על מנת לעמוד בדרישות אלו. **שימו לב**, אין לחרוג מ50 שורות עבור כל פונקציה שהיא, **כולל** פונקציית ה-main.

6.6 קלט

ישנם שני סוגים של קלט שהתוכנה יכולה לקבל:

1) התוכנית תקבל דרך ממשק שורת הפקודה (Command Line Interface או בקיצור CLI) ארבעה ארגומנטים:

- command - הפקודה שרוצים לבצע. הערך יהיה מסוג מחרוזת, כאשר ערכי המחרוזת החוקיים יהיו רק "cipher" ו-"decipher" (עוד על בדיקות תקינות, בהמשך).
- k- מספר ההזחות המבוקש (להצפנה/לפענוח), כך ש- $k \in Z$.
- נתיב לקובץ קלט – בקובץ זה יהיה את הטקסט שהמשתמש מבקש להצפין או לפענח.
- נתיב לקובץ פלט – אל קובץ זה נכתוב את הטקסט לאחר הביצוע של ההצפנה או הפענוח.

2) התוכנית תקבל דרך ה-CLI ארגומנט אחד:

- המחרוזת "test" (הסבר על הסוג הזה בהמשך המסמך).

6.6.1 קריאת הקלט ובדיקות תקינות

שימו לב לנקודות הבאות הנוגעות לקריאת הקלט:

- נזכיר שתוכלו לגשת לארגומנטים שהתקבלו מה-CLI באמצעות argc, argv.
- לא ניתן לבצע השוואה בין מחרוזות באמצעות אופרטור ההשוואה (כלומר ==). כדי לבצע השוואה, תוכלו להשתמש בפונקציית הספריה strcmp(). כדי להשתמש בפונקציה זו עליכם לכלול בראש התוכנית שלכם את הפקודה: #include<string.h>
  הסבר על הפונקציה נמצא כאן.
- בשביל להמיר מחרוזת למספר ניתן להשתמש ב- strtol() . דוגמא לשימוש:

```
char str[] = "2030300";
long ret = strtol(str, NULL, 10);
```

  הסבר על הפונקציה נמצא כאן.
- **~~אנו לא ממליצים~~ כן ממליצים!! (בעקבות הפרסום המוקדם)** לבדוק קלט בעזרת מעבר תו תו (ובדיקה של סוף של מחרוזת, שימו לב שאתם יודעים איך לעשות זאת)

<u>כמו כן, שימו לב להנחות הבאות על הקלט (יש גם בעמוד הבא הנחות!):</u>

- **אינכם רשאים** להניח כי כמות הפרמטרים שתקבלו תקינה (כלומר שלא קיבלתם פחות ארגומנטים מהנדרש, או לחלופין – יותר ארגומנטים מהנדרש).
- **אינכם רשאים** להניח כי הפקודה (הארגומנט הראשון command) שקיבלתם אכן חוקית. כלומר לא ניתן להניח ש- command="cipher" או command="decipher".
- **אינכם רשאים** להניח כי k אכן יהיה מספר שלם, כלומר יתכן שתקבלו גם ערך שאינו מספר או שהוא מספר עשרוני. (שני המקרים נחשבים כקלט לא תקין)

- **ניתן להניח** כי לא תקבלו ב-k מספר מהצורה "03". (קודם 0 או מספר אפסים ואז המספר הרלוונטי)
- **אינכם רשאים** להניח דבר על הטקסט שקיבלתם (דרך הנתיב לקובץ הקלט). בפרט, אינכם יכולים להניח כי הטקסט אינו כולל אותיות שאינן באלפבית' האנגלי, שהטקסט אינו ריק וכדומה.
- **ניתן להניח** כי אורך כל שורה בקובץ הקלט אינו עולה על 1024 תווים.
- **לא ניתן להניח** שהנתיב שקיבלתם לקובץ **הפלט** הוא של קובץ קיים. אם הוא קיים- **יש לדרוס** את הקובץ הקודם. אם הוא לא קיים יש לייצר קובץ חדש (ובשני המקרים לכתוב לתוכו את הטקסט לאחר ההצפנה/הקידוד כמובן).
- **לא ניתן להניח** שקובץ הקלט שתקבלו יהיה תקין או שתצליחו לפתוח אותו.
- **אינכם רשאים** להניח כי תקבלו את המחרוזת "test" כארגומנט אם מספר הארגומנטים הינו 1.

6.6.2 טיפול בשגיאות

במקרים של שגיאה, עליכם להדפיס את המחרוזת הרלוונטית מהרשימה שלהלן ל-stderr ולצאת באופן מיידי מהתוכנית עם קוד שגיאה (להחזיר EXIT_FAILURE) .

חשוב!! שימו לב שעליכם לוודא שאתם סוגרים את הקבצים הפתוחים לפני היציאה מהתוכנית!

הערה: stderr הינו מזהה קובץ ייעודי להדפסת פלט שגיאה. על מנת להדפיס בו הודעות, יש לייבא את stdio.h ואחר כך להשתמש ב-fprintf בצורה הבאה:
fprintf(stderr, "invalid command"), וכך להדפיס את ההודעה invalid command ל-stderr.

- אם כמות הארגומנטים שסופקה לתוכנית אינה תקינה, עליכם להדפיס את המחרוזת הבאה:

```
"The program receives 1 or 4 arguments only.\n"
```

- אם קיבלתם ארגומנט אחד אבל אינו "test", עליכם להדפיס את המחרוזת הבאה:

```
"Usage: cipher test\n"
```

- אם קיבלתם 4 ארגומנטים אך הפקודה שקיבלתם, ארגומנט ה-command, אינה תקינה, עליכם להדפיס את המחרוזת:

```
"The given command is invalid.\n"
```

- אם קיבלתם 4 ארגומנטים אך ערך ה-k שקיבלתם אינו תקין, עליכם להדפיס את המחרוזת:

```
"The given shift value is invalid.\n"
```

- אם קיבלתם 4 ארגומנטים אך יש שגיאה עם אחד הקבצים (קובץ הקלט לא קיים/פתיחת הקובץ נכשלה), עליכם להדפיס את הפקודה:

"The given file is invalid.\n"

במידה ויש כמה שגיאות, צריך להדפיס ל-stderr את ההודעה הראשונה שמקבלים לפי סדר החשיבות הבא-

1. כמות ארגומנטים אינה תקינה.
2. ארגומנט ה-test לא תקין - **רק במקרה שהתוכנה קיבלה בדיוק ארגומנט אחד**, אחרת מדלגים על 2.
3. פקודת command אינה תקינה.
4. ערך הזחה-k לא תקין.
5. בעיה עם נתיב/פתיחת קובץ הקלט/הפלט.

יש להשתמש במאקרו (קבוע) עבור כל אחד מהמחרוזות של השגיאה!

למשל: #define TEST_ERR "Usage: cipher test.\n"

6.7 פלט

תוכנת ה-cipher שלנו תצפין ותפענח רק תווים השייכים לאלפבית האנגלי. כל תו שאינו אות, יישמר כפי שהוא בפלט המוצפן. במילים אחרות, נגדיר $\Sigma = \{'A', 'B', \ldots, 'Z'\} \cup \{'a', 'b', \ldots, 'z'\}$.

עתה, בהנחה שלא היו שגיאות (כמפורט לעיל) התוכנה תפעל כך:

- אם התוכנה קיבלה קלט מסוג 1 (כלומר 4 ארגומנטים):
  - אם הפקודה שהתקבלה היא cipher: התוכנית תכתוב אל קובץ הפלט את ההצפנה של המחרוזת שהתקבלה, באמצעות האלגוריתם שהוצג לעיל באשר לפונקציה cipher ואותה בלבד (כלומר אין לכתוב אל תוך קובץ הפלט תוכן נוסף). יש לצאת מהתוכנה עם קוד הצלחה (להחזיר EXIT_SUCCESS).
  - אם הפקודה שהתקבלה היא decipher: התוכנית תכתוב אל קובץ הפלט את הפענוח של המחרוזת שהתקבלה, באמצעות האלגוריתם שהוצג לעיל באשר לפונקציה decipher ואותו בלבד (כלומר אין לכתוב אל תוך קובץ הפלט תוכן נוסף). יש לצאת מהתוכנה עם קוד הצלחה (להחזיר EXIT_SUCCESS).

- אם התוכנה קיבלה קלט מסוג 2 (כלומר את הארגומנט "test"):
  - יש לצאת מהתוכנית עם קוד יציאה באופן הבא:
    - אם לפחות אחד מהטסטים **נכשל**, קוד היציאה יהיה EXIT_FAILURE.
    - אם **כל הטסטים עברו בהצלחה**, קוד היציאה יהיה EXIT_SUCCESS.

  פירוט נוסף בנושא הטסטים בהמשך המסמך.

זכרו שבכל יציאה מהתוכנית – חשוב להקפיד שכל הקבצים שנפתחו בה יהיו סגורים!

6.8 דגשים והנחיות נוספות:

- נדגיש שוב כי כל אות בטקסט שאינה מופיעה בין 'A' ל-'Z' או בין 'a' ל-'z', תועתק כפי שהיא.
- הזחות על אותיות קטנות ישארו תמיד ב"מעגל" של האותיות הקטנות, והזחות על אותיות גדולות יישארו תמיד ב"מעגל" של האותיות הגדולות. למשל, אם מצפינים (cipher) את האות 'Z' עם k=1, נקבל 'A'.

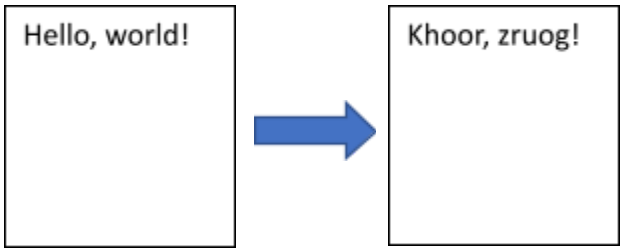
- שימו לב: וודאו שאתם מבינים כיצד פקודת המודולו (השארית) מתנהגת בשפת C עבור מספרים שליליים. הסבר קצר כאן.
- הנכם רשאים ליצור פונקציות עזר כראות עינכם.
- מותר ומומלץ להשתמש ב-stdio.h, stdlib.h, ctype.h, string.h.
- **זכרו להשתמש בקבועים** ולהימנע מהשימוש במשתנים גלובאלים. הגדירו את הקבועים באמצעות מאקרו.
- התוכנה עובדת לפי סוג הקלט (הארגומנט/ים המסופקים).

6.9 דוגמה

- נפתח בדוגמה המדגימה אופן פעולת התוכנה עבור קלט מסוג 1:

התוכנית מקודדת את הטקסט "Hello, world!" שנמצא בקובץ in.txt עבור פרמטר הזחה k=3: וכותבת את הפלט אל הקובץ out.txt באמצעות הפקודה הבאה בטרמינל:

```
$ ./cipher cipher 3 in.txt out.txt
```



in.txt out.txt

עתה, אם נרצה לפענח את הטקסט בקובץ out.txt ("Khoor, zruog!") ולכתוב את הפענוח אל in.txt נוכל להריץ את התוכנית באמצעות הפקודה הבאה:

```
$ ./cipher decipher 3 out.txt in.txt
```

כאשר סימן ה-'$' מסמן פקודה המבוצעת בשורת הפקודה (ב- Terminal), ו- cipher הינה התוכנית הנוצרת על ידי פקודת הקימפול:

```
gcc -Wextra -Wall -Wvla -std=c99 -lm cipher.c tests.c main.c -o cipher
```

- נשתמש בפקודה הבאה כדי להריץ את התוכנה עבור קלט מסוג 2:

```
$ ./cipher test
```

שתחזיר EXIT_SUCCESS במקרה הצלחת כל הטסטים או שתחזיר EXIT_FAILURE במקרה כישלון של לפחות טסט אחד.

הערה: כדי לראות את קוד היציאה של התוכנית בטרמינל אחרי שמריצים אותה, ניתן לכתוב את הפקודה הבאה מיד אחרי הרצת התוכנה: $? echo. ניתן לראות דוגמה כאן.

7 נהלי הגשה

הערה: בעת הגשת התרגיל הקובץ היחידי שאמור להכיל פונקציית main() הוא main.c.
כל קובץ אחר המכיל פונקציית main() ייגרום לכישלון בכל הטסטים!

- אתם נדרשים לקרוא את מסמך נהלי הגשת התרגילים המופיע במודל. לא יהיה ניתן לערער במקרה של אי עמידה בנהלים אלה, ובמקרים מסוימים, הדבר עלול להוביל לציון 0 בתרגיל.
- כמו את כל התרגילים בקורס, את התרגיל יש להגיש דרך הגיט בעזרת הפקודה: git submit.
  ניתן לבצע פעולה זו מספר בלתי מוגבל של פעמים עד לתאריך ההגשה.
- הקפידו להוסיף לגיט רק את הקבצים להגשה.
- כתבו את כל ההודעות שבהוראות התרגיל בעצמכם. <u>העתקת ההודעות מהקובץ עלולה להוסיף תווים מיותרים ולפגוע בבדיקה האוטומטית, המנקדת את עבודתכם</u>.
- זכרו לבצע בקוד שלכם חלוקה הגיונית לפונקציות. בפרט, **הקפידו שהאורך של כל אחת מהפונקציות שתכתבו לא יעלה על 50 שורות.** תהיה הורדת נקודות על פונקציות ארוכות יותר מאורך זה.
- בשפת C ישנן פונקציות רבות העשויות להקל על עבודתכם. **לפני תחילת העבודה על התרגיל, מומלץ לחפש באינטרנט את הפונקציות המתאימות ביותר לפתרון התרגיל.** ודאו שכל הפונקציות שבהן אתם משתמשים מתאימות לתקינה C99 לכל המאוחר (לא C11), וכי אתם יודעים כיצד הן מתנהגות בכל סיטואציה.
- הקבצים להגשה הם: cipher.h, cipher.c, tests.h, tests.c, main.c. **אסור להגיש קבצים אחרים!**
- **בנוסף לפונקציות הרגילות שאסור להשתמש בהן, אסור להשתמש ב- exit(), scanf() או fscanf(). השתמשו בפתרונות אחרים ובפונקציות בטוחות במקום.**
- קובץ להרצה עם פתרון בית הספר זמין לשימושכם בנתיב-

~labcc2/school_solution/ex1/schoolSolution

- דוגמה לשימוש בפתרון בית הספר (עבור קלט מסוג 1):

```
~labcc2/school_solution/ex1/schoolSolution cipher 2 in.txt out.txt
```

- הפקודה הבאה מיועדת רק לבדיקת התרגיל ואינה קשורה להגשה:
  - על מנת לקמפל את הקוד שלכם לתוכנית ברת-הרצה בשם cipher, תוכלו להשתמש בפקודה הבאה:

```
gcc -Wextra -Wall -Wvla -std=c99 -lm cipher.c tests.c main.c -o cipher
```

- **זיכרו כי בקבצי ההגשה, יש לוודא שפונקציית main() תהיה מוכלת אך ורק בקובץ main.c.**
- **שימו לב שהבדיקות מתבצעות על מחשבי בית הספר, לכן וודאו שכל דרישות התרגיל מתקיימות כראוי כאשר הקוד שלכם מקומפל על מחשבים אלו לפני ההגשה.**
- כחלק מהבדיקה האוטומטית תיבדקו על סגנון כתיבת קוד. תוכלו להריץ בעצמכם בדיקה אוטומטית לסגנון הקוד בעזרת הרצה של ה-presubmit.

- אנא וודאו כי התרגיל שלכם עובר את ה- Pre-submission Script ללא שגיאות או אזהרות. קובץ ה-Pre-submission Script זמין בנתיב הבא. **שימו לב שבדיקת ה-presubmit עובדת עבור קבצי tar שמכילים את הקבצים הנדרשים.**

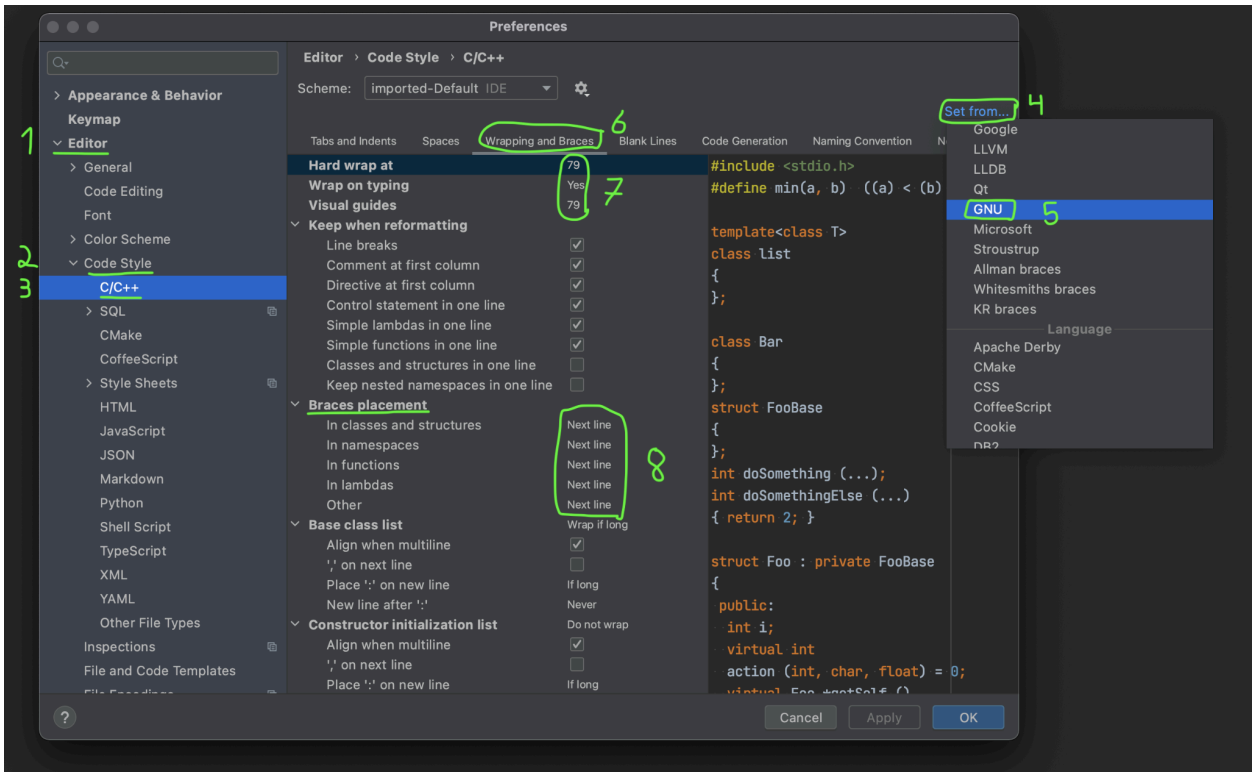
```
~labcc2/presubmit/ex1/run <path_to_tar_file>
```

בהצלחה!!

נספחים

Coding Style

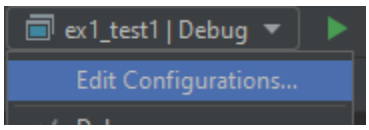
אם תעברו על הצעדים 1 עד 8 (בסדר עולה) שמופיעים בתמונה הבאה, תוכלו להטמיע את ה-coding style הדרוש מכם בהגדרות ה-CLion:



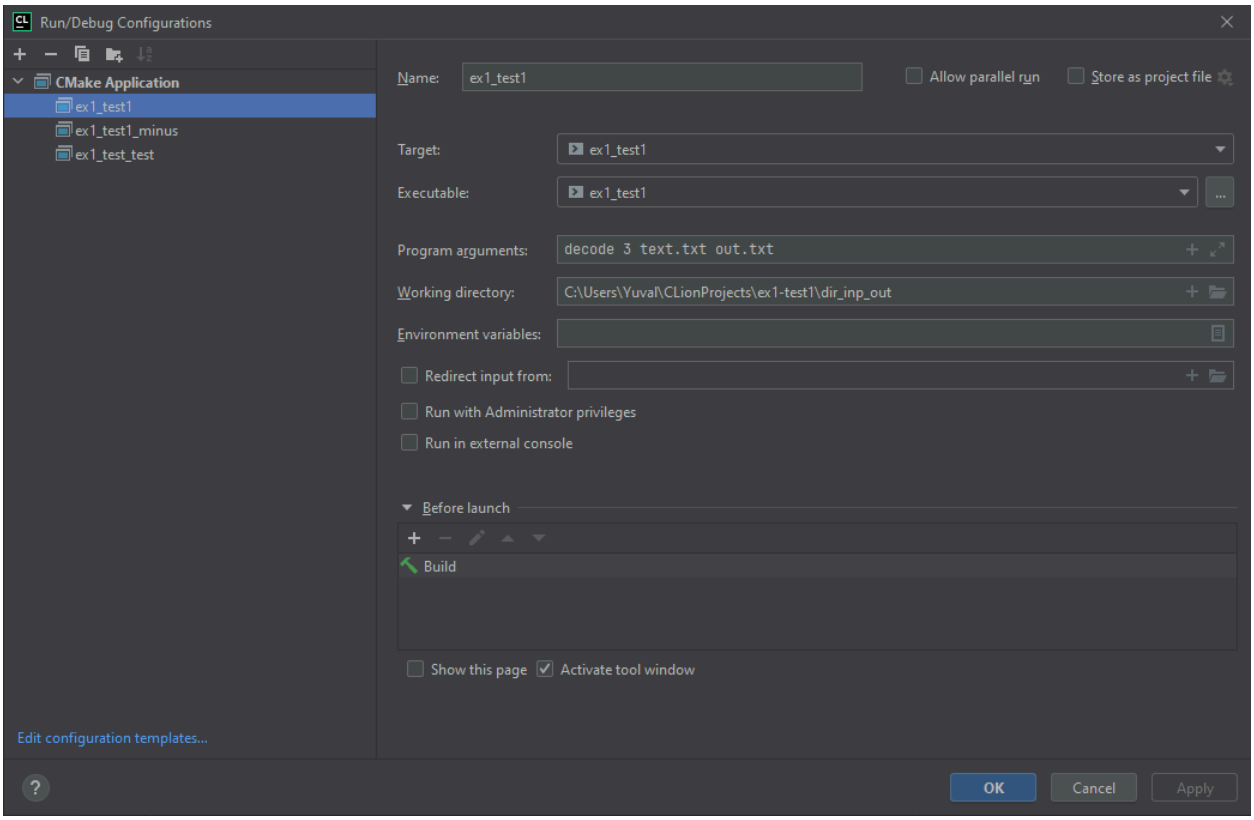
הערה: כדי לעשות reformat לקוד בקובץ כלשהו ב-Clion כדי שהוא יהיה בפורמט ה-Coding style הדרוש, תוכלו (אחרי שביצעתם את שלבים 1-8) ללחוץ על ctrl + alt + L.

Running configurations

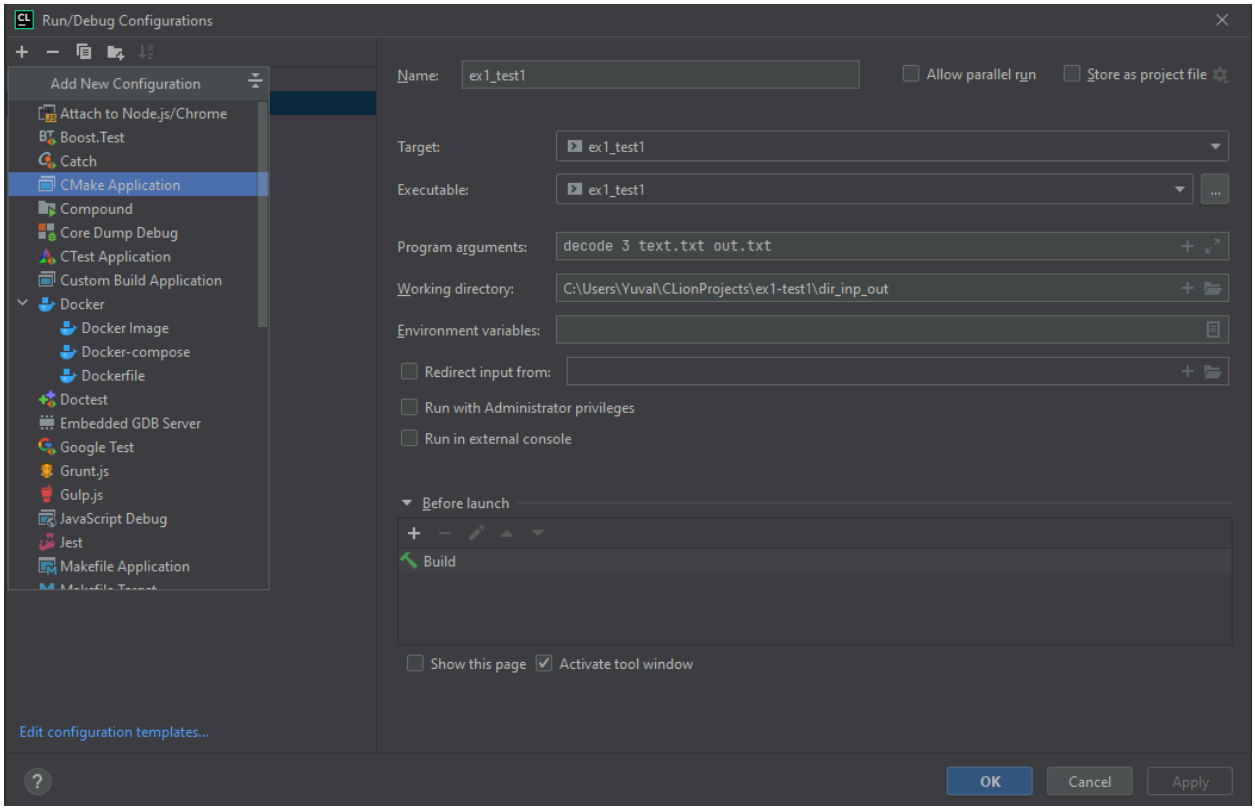
תוכלו להגדיר לעצמכם פרמטרים לריצה אשר יתקבלו כקלט! הדרך לעשות זאת היא ללחוץ על edit configurations:



ואז ייפתח החלון הבא, תחת program arguments תוכלו לכתוב את הארגומנטים של התוכנה (תזכרו, התוכנה תמיד מקבלת גם ארגומנט של מיקום). בנוסף לכך ממליץ לשים ב-working directory את התיקייה שאתם עובדים בה, אחרת הוא יינסה להוציא את קובץ הטקסט מתוך התיקייה cmake-build-debug.



כעת, אם נרצה להוסיף הרצות שונות שנוכל לוודא (למשל הרצה של decipher והרצה של cipher) נוכל לעשות זאת בעזרת ה+ שנמצא בפינה השמאלית העליונה:



נוכל לבחור את השם של ההרצה הנוכחית ולהריץ את הפרמטרים הרלוונטים עבורנו.

וזהו לגבי running configurations!

צופן קיסר – הסבר של שנים קודמות

נתחיל עם כמה הגדרות שישמשו אותנו לאורך מסמך זה:

- אלפבית (א"ב) – היא קבוצה סופית (לרוב של סימנים). מסומנת לרוב ב- Σ.
- ערך הזחה - מספר $k \in Z$ אשר יהיה המפתח של ההצפנה והפענוח (נסביר איך הוא עובד עוד מעט).
- עבור מחרוזת s באורך n, נסמן $s = c_0 c_1 \cdots c_{n-1}$.

הצופן מורכב מ-3 רכיבים:

מפתח הצפנה/פענוח: מספר שלם k.

מצפין: מערכת אשר יודעת לקבל מפתח k ומחרוזת s ולהצפין אותה.

מפענח: מערכת אשר יודעת לקבל מפתח הצפנה k ומחרוזת מוצפנת s ויודעת לפענח אותה.

דרך הפעולה של צופן קיסר:

נסמן ב Σ את האלפבית שה"מצפין" יודע לקודד. המצפין מקבל מחרוזת כלשהי, s, וערך הזחה, k עבור כל תו c_i במחרוזת s **אם** $c_i \in \Sigma$ המצפין יבצע "הזחה ימינה" של c_i, k פעמים. למשל, אם k=2 וקיבלנו את התו 'A' אזי נזיח אותו פעמיים – פעם ראשונה ל-'B' ופעם שניה ל- 'C'. הערך 'C' הוא הערך שמתקבל, אפוא, מהצפנת התו 'A' עם k=2.

עתה, ננסה להיות קצת יותר פורמלים, ונגדיר את צופן קיסר באופן הבא:
זו הגדרה מצומצמת יותר מההגדרה המלאה של צופן קיסר, אך היא תשרת אותנו נאמנה בתרגיל זה. למתעניינים, ראו https://en.wikipedia.org/wiki/Caesar_cipher.

- יהי אלפבית Σ. תהי cipher פונקציה המקבלת 2 פרמטרים: מחרוזת לקידוד (הצפנה), שנסמנה – s וערך הזחה $k \in Z$. הפונקציה cipher מצפינה את s על ידי כך שעבור כל c_i המקיים $c_i \in \Sigma$ היא מבצעת k הזחות מעגליות (cyclic shifts) ימינה (אם k שלילי אז נבצע k הזחות מעגליות שמאלה).
  במילים אחרות, cipher "דוחפת" ימינה k פעמים כל אות אלפביתית ב-s.
  - לדוגמה, עבור $\Sigma = \{'A', 'B', 'C'\}$:
    אם k=1, אזי $'A' \mapsto 'B',\ 'B' \mapsto 'C'$.
    אם k=2, אזי $'A' \mapsto 'C',\ 'B' \mapsto 'A'$ (B הפכה ל-A לאור תכונת המעגליות).

- יהי אלפבית Σ. תהי decipher פונקציה המקבלת 2 פרמטרים: מחרוזת לפענוח, שנסמנה – s וערך הזחה $k \in Z$. הפונקציה decipher מפענחת את s על ידי כך שעבור כל c_i המקיים $c_i \in \Sigma$ היא מבצעת k הזחות מעגליות שמאלה (אם k שלילי אז נבצע k הזחות מעגליות ימינה).
  במילים אחרות, decipher "דוחפת" שמאלה k פעמים כל אות אלפביתית ב-s.

- לדוגמה, עבור $\Sigma = \{'A', 'B', 'C'\}$:

אם k=1, אזי $'B' \mapsto 'A',\ 'C' \mapsto 'B'$.

אם k=2, אזי $'B' \mapsto 'C',\ 'C' \mapsto 'A'$ (B הפכה ל-C לאור תכונת המעגליות).

הערה: עבור שתי הפונקציות שתיארנו, אם $c_i \notin \Sigma$ התו נשאר כמו שהוא אחרי ההצפנה/הפענוח. למשל:

cipher $('D',\ 1)$ = cipher $('D',\ 1)$ = $'D'$.

כפועל יוצא מההגדרות הנ"ל, תהי s מחרוזת ו- $k \in Z$ ערך הזחה, אזי נקבל:

$$s = decipher(cipher\ (s, k), k)$$