האוניברסיטה העברית בירושלים

בית הספר להנדסה ולמדעי המחשב ע"ש רחל וסלים בנין

סדנאות תכנות בשפת C ו־C++ – קיץ (קורס 67320)
C++ – תרגיל 1

תאריך ההגשה של התרגיל והבוחן התיאורטי: יום שלישי, ה־2 בספטמבר, 2019 – עד השעה 23:55;

הגשה מאוחרת (בהפחתת 10 נקודות): יום רביעי, ה־3 בספטמבר, 2019 – עד השעה 23:55.

נושאי התרגיל: **היכרות עם השפה, מחלקות, references, const & const return, types, operators overloading.**

1 רקע

בתרגיל זה נכתוב רכיב אחד מתוכנית שמטרתה לפרוץ את הצפנת ה־RSA (כאשר המספרים קטנים, באופן יחסי, כמובן). נעשה זאת על ידי מימוש של מחלקה המייצגת שדה סופי, $\mathbb{F}_p$, ואלגוריתם המזהה את הגורמים הראשונים המרכיבים את $a \in \mathbb{F}_p$.

2 הגדרות

טרם ניגש לתרגיל, נציג להלן מספר הגדרות ומשפטים מתורת המספרים (שנלמדו בקורסים שונים, כגון אלגברה לינארית (2) ותורת המספרים האלמנטרית) שיעזרו בהבנת הנושא.

מספרים שלמים:

- **הגדרה:** יהי $p \in \mathbb{Z}$. p יקרא **אי־פריק** אם $p \neq \pm 1 \wedge p \neq 0$ וגם קיימים $a, b \in \mathbb{Z}$ כך ש־$p = a \cdot b$ גורר $a = \pm 1$ או $b = \pm 1$.
- **משפט:** כל $1 < n \in \mathbb{N}$ ניתן להצגה כמכפלה של אי פריקים חיוביים.
- **הגדרה:** יהי $p \in \mathbb{Z}$. p נקרא **מספר ראשוני** כעבור $p \neq \pm 1 \wedge p \neq 0$, קיימים $a, b \in \mathbb{Z}$ כך ש־$p | ab$ – אזי $p | a$ או $p | b$ (הסימון $a | b$ משמעו ש־a מחלק את b).
- **משפט אוקלידס:** יש אינסוף מספרים ראשוניים.

- **משפט:**יהיו $a,b\in\mathbb{Z}$ שונים מ־0. נאמר ש־$d\in\mathbb{Z}$ הוא **מחלק משותף** של a ו־b אם $d|a$ וגם $d|b$ (ללא שארית).

- **משפט:** יהיו $a,b\in\mathbb{Z}$ שונים מ־0. קיים $d\in\mathbb{Z}$ יחיד כך שכל מחלק משותף של a ו־b מחלק גם את d. במקרה זה, d יקרא **המחלק המשותף המקסימלי** (Greatest Common Divisor או בקצרה GCD) של a ו־b ונסמנו $d=gcd(a,b)$.

- **למת Bezout:** יהיו $a,b\in\mathbb{Z}$ שאינם 0. ישנם $p,q\in\mathbb{Z}$ כך ש־$p\cdot a+q\cdot b=gcd(a,b)$.

- **הגדרה:** נאמר ש־a ו־b זרים אם $gcd(a,b)=1$.

- **משפט:** יהיו $a\in\mathbb{N},p\in\mathbb{Z}$. אם p אי־פריק אזי $p|a$ או $gcd(a,p)=1$.

- **משפט:** יהי $1\neq p\in\mathbb{N}$. p ראשוני אם ורק אם p אי־פריק.

- **המשפט היסודי של האריתמטיקה:** ניתן להציג כל מספר טבעי שונה מ־1 כמכפלה ייחודית של מספרים ראשוניים, עד כדי שינוי הסדר של הגורמים. במילים אחרות, יהי $1<n\in\mathbb{N}$. קיימת סדרה של מספרים ראשונים, $(p_1,\ldots,p_r)$, סדרה של מספרים טבעיים, $(n_1,\ldots,n_r)$ – כך שהם מקיימים יחדיו $n=p_1^{n_1}\cdot\ldots\cdot p_r^{n_r}=\prod_{k=1}^{r}p_k^{n_k}$.

שדות סופיים:

- **משפט:** לכל $a,b\in\mathbb{Z}$ קיים מחלק משותף מקסימלי d יחיד עד כדי סימן.ס**הגדרה:** יהי $p\in\mathbb{Z}$ מספר ראשוני. נאמר שהשדה $\mathbb{F}_p$ הוא **שדה סופי** (או שדה גלואה, Galois Field) כאשר יש בו מספר סופי של אברים. כלומר, אם $\underbrace{1+1+\cdots+1}_{p\ \ times}=0$ ואז
$p=char(\mathbb{F}_p)$.
מסקנה 1: השדה הסופי $\mathbb{F}_p$ מקיים $|\mathbb{F}_p|=p^n$ עבור n כלשהו (כי $\mathbb{F}_p$ הוא מ"ו).
מסקנה 2: על השדה הסופי $\mathbb{F}_p$ מוגדרות היטב פעולות חיבור, חיסור, כפל וחילוק (למעט חלוקה באפס). יתרה מכך, פעולות אלו מקיימות את אקסיומות השדה.

- **משפט:** יהי $p\in\mathbb{Z}$ מספר ראשוני. עבור כל $n\in\mathbb{N}$ קיים שדה מסדר p^n. שדה זה יחיד עד כדי איזומורפיזם.

- **משפט:** הפונקציה $\varphi(n)$ קרויה **פונקציית אוילר** ושווה לכמות המספרים הטבעיים הזרים ל־n ואינם גדולים ממנו. במילים אחרות, $\varphi(n)$ שווה לכמות המספרים השלמים k בטווח $1\leq k\leq n$, שעבורם המחלק המשותף הגדול ביותר, $gcd(k,n)$, שווה ל־1.

3 הצפנת RSA

תחילה, נציין כשרקע לנושא הצפנת RSA (מהי הצפנת RSA ב־"שפת רחוב", למה צריך אותה וכו') – מצורף לתרגיל כנספח 1 סיפור רקע. מומלץ בחום לסטודנטים שאינם מכירים את הנושא לעצור כעת ולקרוא את הסיפור (הנאה ומלודרמה מובטחים ☺).[1]
עתה, נבחן בקצרה את עקרונותיה של RSA והקשר בינה ובין מספרים ראשונים ופירוק לגורמים. ננסה לתאר בקצרה, מבלי להיכנס לפרטים רבים מדי, את שלבי יצירת המפתח הפרטי והמפתח הפומבי:

[1]בנוסף לסיפור הרקע, כדי לקבל אינטואיציה ניתן לצפות בסרטון הזה (שממחיש את הנושא באמצעות צבעים): https://www.youtube.com/watch?v=3QnD2c4Xovk. סרטון מגניב, מבטיחים!

1. יהיו $p, q \in \mathbb{Z}$ שני מספרים ראשוניים **גדולים.** נסמן $n = pq$ ומעתה ואילך נבצע את כל החישובים מעל השדה $\mathbb{Z}_n$.
2. **למפתח פומבי,** נבחר $p_{pub} \in \mathbb{Z}_n$ שמקיים $1 < p_{pub} < \varphi(n)$ וגם ש־p_{pub} זר ל־$\varphi(n)$.
3. **למפתח פרטי,** נבחר $p_{priv} \in \mathbb{Z}_n$ שהוא ההופכי הכפלי המודולרי של $p_{pub} \ \ (mod\, \varphi(n))$, כלומר שהוא מקיים: $p_{priv} \cdot p_{pub} \equiv 1 \ \ (mod\, \varphi(n))$.

עתה, נוכל לפעול כך:

1. **הצפנה:** נניח ש־$m \in \mathbb{Z}_n$ מייצג את התוכן שנרצה לשלוח. נשתמש במפתח **הפומבי** של **הנמען** ונקבל $c \equiv m^{p_{pub}}\,(mod\, n)$. את c (קיצור ל־$cipher\, text$) נשלח לנמען.
2. **קידוד**: הנמען רוצה לשחזר מ־c את m. הוא יכול לעשות זאת בעזרת המפתח **הפרטי** שלו, כך: $c^{p_{priv}} \equiv (m^{p_{pub}})^{p_{priv}} \equiv m\,(mod\, n)$.

אם כך, נוכחנו לראות כי באמצעות שימוש באותם מספרים ראשוניים – p ו־q יכולנו ליצור שני מפתחות, אחד המשמש להצפנת מסרים עבור נמען ספציפי (המפתח הפומבי) ואחד המשמש לקידוד המסרים (המפתח הפרטי). אם כך, RSA מסתמך במובלע על שני עקרונות חשובים:

- המסקנה שהוצגה לעיל מהמשפט היסודי של האריתמטיקה – לכל מספר טבעי גדול מ־1 ניתן להצגה כמכפלה של ראשוניים חיוביים יחידה עד כדי סדר המחוברים.
- בעיה מוכרת בתורת המספרים – לפיה לא קיימת שיטה טובה מספיק למצוא את הפירוק הראשוני של מספר גדול מספיק. הרי אם נצליח לבצע פירוק לראשונים, נוכל לחשוף את המפתח הפרטי.[2]

משכל זאת נאמר – נוכל לגשת למטלה שלנו. בתרגיל זה נרצה לעזור להאקרית המחוננת, איב, לפרוץ את הצפנת ה־RSA ולחשוף את סודותיה של אליס. נעשה זאת באמצעות מימוש מחלקה המייצגת שדה גלואה (לחשבון מודולרי) וכן מספר אלגוריתמים להם איב זקוקה לפיצוח החידה – כגון gcd ופירוק לגורמים ראשונים.

לסיום, נציין שהאלגוריתמים שנממש בתרגיל זה יצליחו לפצח מספרים בסדר גודל של 10^7 ואף 10^{10}. אלא, שהקושי "בעולם האמיתי" הוא ש־RSA מסתמך על מספרים גדולים הרבה יותר. למשל, זהו מספר RSA עם 2048 בייטים שהוגרל לפי האלגוריתם שתואר לעיל:

$$
\begin{aligned}
N = {} & 22195859722513341975692363967031807897501687649650297564038938974\\
& 40472973300869166129573528370182119818888138862560715664729475741 65\\
& 82905447852527092034712972471000265683649025961474360927320887485 96\\
& 27487248804229163610178140682635652822589238663020190549283972583 56\\
& 99585369843623379165197501840824868872352316595460884766587047882 51\\
& 19099841347521045216025366997048810814735935650695172423646256072 56\\
& 44560071132660318101243797309460459557694660447867492708909510181 65\\
& 77952862837053675721195794477804987257801395089905388758835405552 88\\
& 91688188128566411430554451159860786361022005446500087736267895062 78\\
& 3487909314684563
\end{aligned}
$$

[2] במובלע יש כאן הישענות על בעית הלוגריתם הדיסקרטי – לפיה טרם נמצא אלגוריתם יעיל מספיק לחישוב לוגריתמים דיסקרטים באופן כללי.

מציאת המספרים הראשוניים המרכיבים את N, עם הכלים המתמטים והמשאבים הזמינים היום, תהא משימה שתיקח אלפי שנים ואף יותר – גם אם היה בידינו את כל כוח החישוב בעולם. אגב, בעת האחרונה אף החלו להשתמש במספרים **באורך 4096 בייטים** – דהיינו כפול ביטים מהמספר שלעיל. אלגוריתם אחד המתעתד לייעל הליך הפירוק לגורמים הוא האלגוריתם של Shor, אך הוא דורש שימוש במחשב קוונטי. [3]

4 המחלקות המרכיבות את התרגיל

4.1 המחלקה GField

נפתח את תרגיל זה עם המחלקה GField, המייצגת ממספר בשדה גלואה ממאפיין $p \in \mathbb{N} \wedge 2 \leq p$ ודרגה $l \in \mathbb{N} \cup \{0\}$ (קרי, זה השדה $\mathbb{Z}_{p^l}$). על המחלקה לתמוך ב־API הבא:

	<u>תיאור</u>	**<u>הערות</u>**
פעולות מחזור החיים של האוביקט		
בנאי ברירת מחדל	בנאי ברירת מחדל המאתחל את השדה עם $p = 2,\ l = 1$.	
בנאי 1	בנאי המקבל את p ומגדיר $l = 1$.	p מטיפוס `long`.
בנאי 2	בנאי המקבל את p ואת l.	$l,\ p$ מטיפוס `long`.
בנאי העתקה	מימוש של בנאי העתקה.	
destructor	מימוש destructor.	
פעולות		
`getChar`	קבלת מאפיין השדה (p).	
`getDegree`	קבלת סדר השדה (l).	
`getOrder`	קבלת סדר השדה (p^l).	
`isPrime`	פעולה המקבלת מספר, p, ומחזירה אמת אם זהו מספר ראשוני.	p מטיפוס `long`. עליכם לממש פעולה זו ב־$O(\sqrt{n})$. **זו תהיה פעולה <u>סטטית</u>.**
`gcd`	פעולה המקבלת $a,\ b \in \mathbb{Z}_{p^l}$, ומחזירה את $gcd(a,\ b)$.	$a,\ b$ מטיפוס `GFNumber`. ערך החזרה מטיפוס `GFNumber`.
`createNumber`	פעולה המקבלת $k \in \mathbb{Z}$ ומחזירה מופע שלו מתוך השדה.	k מטיפוס `long`. ערך החזרה מסוג `GFNumber`.
אופרטורים		
השמה	תמיכה באופרטור ההשמה (=).	השמה לכל ערכי השדה.
השוואה	תמיכה באופרטורי ההשוואה == ו־=!.	ההשוואה תהא לפי סדר השדה (כלומר לפי p^l).
הדפסה	הדפסה ל־`std::cout` (<<).	ראו פורמט בהמשך.
קריאה	קריאת השדה מ־`std::cin` (>>).	ראו פורמט בהמשך.

דגשים והנחות:

- את המחלקה עליכם להגדיר בקבצים `GField.h, GField.cpp`.

[3] ניתן לקרוא על האלגוריתם של שור כאן: https://interestingengineering.com/how-peter-shors-algorithm-dooms-rsa-encryption-to-failure וכאן https://en.wikipedia.org/wiki/Shor%27s_algorithm/

- פורמט **הפלט** יהיה:

```
GF({char}**{degree})
```

כאשר `{char}` מסמן את p ו־`{degree}` מסמן את l. למשל, עבור $p^l = 2^1$ נדפיס: `GF(2**1)` (שימו לב – אין שורה חדשה בסוף הפלט).

- פורמט **הקלט** הוא:

```
{char} {degree}
```

כש־`{char}` מסמן את p ו־`{degree}` את l. למשל, אם $p^l = 2^1$ נקלוט: 1 2.

4.2 המחלקה GFNumber

עתה, לאחר שמימשנו את מחלקת השדה, ניגש למימוש המחלקה GFNumber, המייצגת מספר כלשהו בשדה גלואה מסדר p –נסמנו $n \in \mathbb{Z}_{p^l}$. על המחלקה לתמוך ב־API הבא:

	התיאור	**הערות**
	פעולות מחזור החיים של האוביקט	
בנאי ברירת מחדל	בנאי שלא מקבל ארגומנטים ומאתחל את המספר 0 בשדה $\mathbb{Z}_{2^1}$.	
בנאי 1	בנאי המקבל את n מהשדה $\mathbb{Z}_{2^1}$.	n מטיפוס `long`.
בנאי 2	בנאי המקבל את n ואוביקט `GField`.	n מטיפוס `long`.
בנאי העתקה	מימוש של בנאי העתקה.	
destructor	מימוש destructor	
	פעולות	
`getNumber`	פעולה המחזירה את n.	המספר שמחוזר יהיה מטיפוס `long`.
`getField`	פעולה המחזירה השדה של n.	פעולה המחזירה את אוביקט מסוג `GField`.
`getPrimeFactors`	הפעולה מחשבת את המספרים הראשונים וחזקותיהם שמרכיבים את n.	ראו את ההערות להלן.
`printFactors`	פעולה המדפיסה **מחרוזת** המתארת הפירוק לגורמים ראשונים של n.	פורמט המחרוזת יהיה כפי שמופיע בהערות לפרק זה.
`getIsPrime`	פעולה המחזירה אמת אם המספר הוא ראשונית ושקר אחרת.	
	אופרטורים	
השמה	תמיכה באופרטור ההשמה ($=$).	השמה לכל ערכי המספר.
חיבור	תמיכה באופרטורים $+, +=$.	
חיסור	תמיכה באופרטורים $-, -=$.	
כפל	תמיכה באופרטורים $*, *=$.	
שארית (מודולו)	תמיכה באופרטורים $\%, \%=$.	
השוואה	תמיכה באופרטורי ההשוואה $==$ ו־$!=$.	ההשוואה מתבצעת על p, n.
יחס	תמיכה ב־$<, <=, >, >=$.	ההשוואה מתבצעת על n.
הדפסה	הדפסה ל־`std::cout` ($>>$).	ראו פורמט בהמשך.
קריאה	קריאת מספר מ־`std::cin` ($<<$).	ראו פורמט בהמשך.

דגשים והנחות:

- את המחלקה עליכם להגדיר בקבצים `GFNumber.h, GFNumber.cpp`.
- בכל האופרטורים האריתמטים שתממשו למחלקה זו, כלומר $-=, +=, +$ וכדומה, **עליכם לתמוך בשתי וריאציות:** הראשונה שמבצעת את הפעולה על שתי GFNumber, והשניה שמבצעת את הפעולה על lparam שהוא GFNumber ו־rparam שהוא long.
- למען פשטות הדברים, יש לאפשר ביצוע פעולות אריתמטיות ופעולות יחס (כלומר של שדה סדור) **אך ורק על איברים באותו השדה. לכן, למעט באופרטורים == ו־=!, אם השדדות שונים – עליכם להציג שגיאה על ידי שימוש** ב־`assert`.
- כזכור, הפעולה `getPrimeFactors` נדרשת למצוא את הגורמים הראשוניים המרכיבים את n. לפיכך, הפעולה תחזיר מערך חד־מימדי **שהוקצה דינמית**, שכולל את כל המספרים הראשוניים שמרכיבים את n – כולם מטיפוס GFNumber. לדוגמה, נניח ש־$n = 297936$. אם כך, הפירוק שלו למספרים ראשונים הוא $n = 2^4 \cdot 3^2 \cdot 20691^1$ ולכן נחזיר מצביע למערך שהוקצה דינמית ונראה כך: `[2, 2, 2, 2, 3, 3, 2069]`.
כמו כן, נגדיר שגודל המערך, טיפוס מסוג `int` שאילו נשלח מצביע, יהיה 7.
זכרו – אין להשתמש ב־`malloc` ו־`realloc` ב־C++. עליכם לדאוג להרחבת גודל המערך בצורה דינמית בהתאם לכמות איבריו. כמו כן, שימו לב שסדר מיקום האיברים במערך אינו משנה.
- הפורמט של המחרוזת שתודפס מהפעולה `printFactors` יהיה:

```
{n}={p_1}*...*{p_r}
```

כאשר `{n}` מייצג את n ו־`{p_1}` עד `{p_r}` מייצגים את סדרת המספרים הראשוניים, $(p_1, \ldots, p_r)$.
דוגמה: נניח ש־$n = 297936$. אם כך, הפירוק שלו למספרים ראשונים הוא $n = 2^4 \cdot 3^2 \cdot 20691^1$ ולכן נדפיס את הפלט הבא:

```
297936=2*2*2*2*3*3*20691
```

שימו לב: סדר ההדפסה אינו משנה. בסוף הפלט תבוא שורה חדשה (כיצד תעשו זאת ב־C++? רמז – יש לזה קבוע). כמו כן, אם המספר ראשוני – יודפס המספר בלבד.

- פורמט **הפלט** יהיה:

```
{n} GF(...)
```

כאשר `{n}` מסמן את n ו־`GF(...)` את $\mathbb{Z}_p^l$. למשל אם נבחר את $4 \in \mathbb{Z}_{2^3}$ אזי נקבל את הפלט: `4 GF(2**3)`.

- פורמט **הקלט** יהיה:

```
{n} {GF}
```

כאשר `{n}` מסמן את n ו־`{GF}` מסמן את פורמט הקליטה $\mathbb{Z}_p^l$. למשל אם נרצה לקלוט את $4 \in \mathbb{Z}_{2^3}$ אזי נזין: `4 2 3`.

- **לא ניתן** להניח שהערכים המספריים יקיימו $0 \leq k < p^l$. עליכם לבצע חשבון מודולו כדי להתאימם (ולא להציג שגיאה במקרה בו $k \geq p^l$).
- זכרו שכל הפעולות נעשות בחשבון מודולרי לפי השדה $\mathbb{Z}_{p^l}$.

4.3 הערות כלליות

- **שימו לב:** ה־API הנ"ל מציג לכם את שמות הפונקציות המחייבות, הפרמטרים, ערכי החזרה וטיפוסיהם. בעת מימוש ה־API, עליכם ליישם את העקרונות שנלמדו בקורס באשר לערכים קבועים (constants) ומשתני ייחוס (references). **שימוש בקונבנציות אלו הוא חלק אינטגרלי מהתרגיל, עליו אתם מקבלים ניקוד.**

- את הפעולה getFactors עליכם לממש לפי אלגוריתם ראו של פולרד (Pollard's rho). **אם האלגוריתם נכשל,** עליכם להשתמש ב־Trail Division. תוכלו למצוא הסבר לאלגוריתמים ו־pseudo code בנספח 2 לתרגיל.

- **ניתן** להניח כי הקריאות שנעשות לפונקציות, אכן נעשות עם טיפוסי הנתונים אליהם אתם נדרשים. **מנגד, לא ניתן** להניח כי הערכים גופם תקינים.

- **לא ניתן** להניח כי הערכים שתקבלו תקינים. בפרט, לא ניתן להניח ש־$l \leq 0$ או ש־p הוא מספר ראשוני. עליכם לוודא זאת.

4.4 טיפול בשגיאות

עליכם לטפל בכל השגיאות האפשריות שלעיל באמצעות פקודות assert.[4] זכרו לייבא את הקובץ cassert (ולא את assert.h, שכן אנו נותנים עדיפות לספריות C++).

5 התוכנית IntegerFactorization

עתה, עליכם לכתוב בקובץ IntegerFactorization.cpp תוכנית העושה שימוש במחלקות שכתבתם. בקצרה, התוכנית תקלוט 2 מספרים משדה $\mathbb{F}_{p^l}$ כלשהו, ותדפיס את מלוא פעולות החשבון שלהם. לאחריהן, היא תדפיס את המספרים הראשוניים מהם המספרים מורכבים.

5.1 קלט

התוכנית תקלוט דרך ה־std::cin 2 מספרים (בפורמט הקליטה של GFNumber).
הנחות על הקלט:

- **כדי להקל על מימוש התרגיל, הפעם ניתן** להניח שנשלח מספר פרמטרים תקין. **מנגד, לא ניתן להניח** שהערכים שנקלטו אכן מסוגי הנתונים התקינים (**רמז:** היעזרו ב־std::cin.fail()).

- **לא ניתן** להניח שתקבלו שני מספרים מאותו השדה. במקרה בו המספרים משדות שונים – **עליכם לנהוג כבמקרי שגיאה.**

אם נתקלתם בשגיאה, עליכם לסגור את התוכנית באופן מיידי עם **קוד שגיאה** (ללא הדפסות).

[4] בהמשך הקורס תראו מתי יש להשתמש ב־assert ומתי יש להשתמש בכלי אחר, אותו ראיתם בקורס "מבוא לתכנות מונחה עצמים" – exceptions. עם זאת, לעת הזו עליכם להשתמש ב־assert בלבד.

5.2 פלט

יהיו $a, b \in \mathbb{F}_{p^l}$. עליכם להדפיס, בפורמט הפלט של GFNumber, את תוצאת הפעולות הבאות (בסדר הבא, ללא כל תחילית, ובצירוף ''\n'' בסוף כל פלט):

- $a + b$;
- $a - b, \;\; b - a$ (כל תוצאה בשורה נפרדת);
- $a \cdot b$;
- המספרים הראשונים שמרכיבים את a (מופרדים ברווח);
- המספרים הראשונים שמרכיבים את b (מופרדים ברווח);

6 דוגמה

להלן נראה דוגמת הרצה של IntegerFactorization:

```
$ ./IntegerFactorization
8 11 2
9 11 2
17 GF(11**2)
120 GF(11**2)
1 GF(11**2)
72 GF(11**2)
2*2*2
3*3
```

(כאשר השורה שנפתחת ב־$ מסמנת את הפקודה שהוקלדה והשורות בירוק מסמנות קלט מהמשתמש).

7 נהלי הגשה

- קראו בקפידה את הוראות תרגיל זה ואת ההנחיות להגשת תרגילים שבאתר הקורס.
- זכרו שהחל מתרגיל זה עליכם לקמפל את התוכנית כנגד מהדר לשפת C++ בתקן שנקבע בקורס. כמו כן, זכרו שעליכם **לתעדף** פונקציות ותכונות של C++ על פני אלו של C. למשל, נעדיף להשתמש ב־new ו־delete על פני malloc ו־free, וכן נעדיף להשתמש ב־std::string מאשר ב־char*.
- **נזכיר:** כאמור בהנחיות הכלליות להגשת תרגילים ־ הקצאת זיכרון דינמית מחייבת את שחרור הזיכרון, למעט במקרים בהם ישנה שגיאה המחייבת סגירת התוכנית באופן מיידי עם קוד שגיאה (כלומר קוד יציאה השונה מ־0). תוכלו להיעזר בתוכנה valgrind כדי לחפש דליפות זיכרון בתוכנית שכתבתם.
- פתרון בית הספר זמין בנתיב:

```
~proglab/www/cpp_ex1/IntegerFactorization
```

- עליכם ליצור קובץ tar הכולל את הקבצים GField.h, GField.cpp, GFNumber.h, GFNumber.cpp, IntegerFactorization.cpp, README. ניתן ליצור קובץ tar כדרוש על ידי הפקודה:

```
$ tar -cvf cpp_ex1.tar <files...>
```

שימו לב: קבצי קוד המקור שתכתבו נדרשים להתקמפל כהלכה עם std++14, כנדרש בהוראות להגשת תרגילים שפורסמו באתר הקורס.

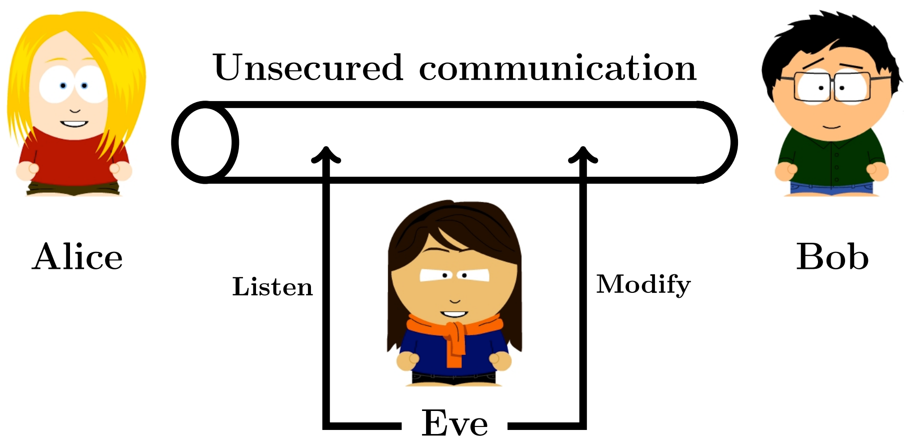
- אנא וודאו כי התרגיל שלכם עובר את ה־Pre-submission Script **ללא שגיאות או אזהרות**. קובץ ה־Pre-submission Script זמין בנתיב.

```
~proglab/www/cpp_ex1/presubmission
```

בהצלחה!!

8 נספח 1 – סיפור הקדמה להצפנת RSA

להלן נציג סיפור קצר (קריאת רשות, כמובן) כדי להציג צורך (אחד) בהצפנת RSA:[5]
היו היה פעם (לא, לא. ☺). אליס ובוב חברים לעט[6], המשוחחים על נושאים אישיים חשובים בחדר צא'ט. יום אחד אליס סיפרה לבוב על אודות אירוע אישי וחשוב בחייה. אך אבוי, איב – האקרית מחוננת ויריבתה המושבעת של אליס – הצליחה להשיג גישה לערוץ התקשורת של אליס ובוב. כך, כל הודעה שמוחלפת בין הצמד עוברת דרך איב, וזו יכולה לקרוא אותה ואף לערוך אותה כראות עיניה. ערוץ התקשורת נראה כך:



איב החליטה לפרסם את וידוייה של אליס ברשת האינטרנט – וזאת לאחר ששלחה לבוב הודעה ערוכה עם מסר שונה בתכלית מהמסר שאותו כתבה אליס במקור. הצמד, שנקלע למשבר ביחסים, גילה בסופו של יום על התרמית. אלא שכעת, אלו חוששים לשתף האחת את השני בנושאים אישיים שמא אלו יקראו ואף יערכו שוב על ידי צד ג'.
מה יכולים אליס ובוב לעשות כדי למנוע מאיב לקרוא את הודעותיהם? הפתרון הוא להשתמש בהצפנת **מפתח פומבי**, המבוססת על אלגוריתם RSA! כך הצמד יפעל: אליס ובוב יבחרו, **כל אחד** מהם, שתי מפתחות – מפתח פומבי ומפתח פרטי. המפתח פומבי משמש להצפנת המסרים, בעוד המפתח הפרטי משמש לקידוד המסרים שהוצפנו על ידי המפתח הפומבי. מפתחות אלו הם בעצם מספרים בהם יעשו שימוש כדי לקודד את הודעתם, אך לכך נגיע מיד. אליס ובוב יפרסמו את המפתח הפומבי שלהם "לציבור הרחב", בעוד שכל אחד ישמור את המפתח הפרטי שלו "בסוד" (גם בוב, למשל, לא ידע מהו המפתח הפרטי של אליס).
כעת, אליס תכתוב את וידויה (המסעיר!) ותצפין אותו באמצעות המפתח הפומבי של בוב – שכזכור, ידוע לכל. עתה, משנשלחה הודעתה של אליס, זו נלכדה על ידי איב – שממתינה ללא הרף להדלפת מידע נוסף. אלא מהי? שכעת ההודעה מוצפנת ולפיכך אין בידי איב לדעת מהו תוכן ההודעה. כל שבידה הוא גיבוב מספרים ולפיכך גם אינה יכולה לערוך את תוכן ההודעה. כעת, ההודעה מועברת לידי בוב. בוב מקבל את אותו "גיבוב המספרים", אך שבידו המפתח הפרטי הסודי שלו. בוב משתמש במפתח הפרטי שלו – ומשחזר באמצעותו את התוכן המקורי של ההודעה. והנה – בוב קורא את הודעתה המקורית של אליס, מבלי שנקראה או נערכה על ידי איב. הם חיו באושר ואושר עד לעצם היום הזה.

[5] כאמור לעיל, ניתן לקבל גם אינטאיציה לרעיון בעזרת צפייה בסרטון הזה, שממחיש את הרעיון בעזרת צבעים (מומלץ!): https://www.youtube.com/watch?v=3QnD2c4Xovk.
[6] חברים לעט: https://www.collinsdictionary.com/dictionary/english/pen-pal.

9 נספח 2 – אלגוריתמים לפירוק לגורמים

להלן נציג מספר שני אלגוריתמים לפתרון התרגיל – ונראה pseudo code שלהם.

9.1 מבוא – אלגוריתם ניסוי ותהייה (Brute force)

ישנם מספר אלגוריתמים יעילים לפירוק לגורמים, למשל אלגוריתם ראו של פולרד שנראה בהמשך, האלגוריתם של דיקסון, האלגוריתם של פרמה (אותם לא נראה בתרגיל זה) וכדומה. טרם נעיין באלגוריתם של פולרד, ננסה להשיב לשני שאלות מקדמיות שעשויות לעלות:

1. השאלה הראשונה, היא מדוע אנחנו זקוקים לאלגוריתם לפירוק לגורמים ראשונים? בהקשר שלנו, התשובה כבר מסתתרת בתוך התרגיל: הצפנת RSA, שעל בסיסה עובדים ישומים רבים ביותר כיום – מבוססת על RSA. למשל, כשאתם כותבים בדפדפן כתובת של אתר שמתחילה ב־https:// – אתם בעצם פותחים ערוץ תקשורת "מאובטח" עם אתר האינרנט אליו אתם פונים, כשהאבטחה היא אבטחת מפתח פומבי, שכאמור מבוססת על RSA. לכן, אם נרצה לפצח את ההצפנה – נצטרך לחשוב על אלגוריתם לפירוק מספרים ראשוניים.

2. השאלה שניה, היא מדוע אנו זקוקים לאלגוריתם מתמטי מיוחד? האם לא ניתן להשתמש באלגוריתם Brute-force נאיבי? טלו למשל את האלגוריתם הבא:

Algorithm 1 Direct Search Factorization

```
procedure DirectSearchFactorization(n):
  factors ← []
  i ← 2
  while i ≤ ⌊√n⌋ then:
    if n (mod i) == 0 then:
      factors.append(i)
      n = n/⌊i⌋
    else:
      i ← i + 1
  if n > 1 then:
    factors.append(n)
  return factors
```

אלגוריתם זה נקרא Direct Search Factorization ומתבסס על שיטה בשם Trail Division. באלגוריתם זה אנו מבצעים באופן סיסטמטי ניסיונות חלוקה כדי לגלות מיהם המחלקים של n. שימו לב שמכיוון שאנו מחפשים את המחלקים של n, בהתאם למשפטים שהובאו במסגרת התרגיל, אין צורך לבדוק מספרים הגדולים מ־$\lfloor\sqrt{n}\rfloor$. לכן, קל לראות שהאלגוריתם פועל בסיבוכיות $O\left(\lfloor\sqrt{n}\rfloor\right)$ (כש־n הוא הקלט).
עם זאת, מדובר באלגוריתם שאינו יעיל כלל כאשר מדובר במספרים גדולים. כך למשל, הרצת אלגוריתם זה על מספרים כגון 10^7 או 10^8 עשויה לקחת דקות עד עשרות דקות, לעומת אלגוריתמים יעילים יותר המגיעיםע לפתרון בתוך חצי־שניה עד שניה. זכרו שכשראינו מספר RSA, דובר במספר עם עשרות ספרות. מהסיבה הזו אנו זקוקים לאלגוריתם יעיל יותר.

9.2 אלגוריתם rho (ρ) של פולרד

בשנת 1975 הציג פולרד אלגוריתם חדש לפירוק מספרים לגורמים. אלגוריתם זה היה בולט מאחר שדורש יחסית מעט משאבי זיכרון (כלומר, יעיל ברמת Space complexity) ופועל בסיבוכיות התלויה **בגורם הראשוני** ולא בקלט. **זהו הרעיון בבסיס האלגוריתם:**
תחילה, יהי $n \in \mathbb{N}$ ונניח בלי הגבלת הכלליות שזהו מספר **חיובי אי זוגי.**[7] מטרתינו היא למצוא $p, q \in \mathbb{N}$ כך ש־$n = p \cdot q \in \mathbb{N}$ ו־p הוא מחלק לא טריוויאלי. נבחר תחילה את $f(x)$ להיות פולינום כלשהו בשדה מודולו n (שדה גלואה $\mathbb{F}_n$) – למשל $f(x) = x^2 + 1$.
כעת, תהי x_k סדרה של מספרים מעל $\mathbb{F}_n$ ומספר $0 \neq c \in \mathbb{F}_n$ **שנגריל רנדומלית** (כלומר $1 \leq c = rand(1, n) < n$). את ערכי x_k נגדיר כך:

$$\forall k \in \mathbb{N} \qquad x_k = \underbrace{(f \circ \ldots \circ f)}_{k\, times}(c)$$

במילים אחרות, האיבר הראשון בסדרה הוא איבר רנדומלי מתוך שדה $\mathbb{F}_n$ ושאר איברי הסדרה הן התוצאה של ההרכבה של $f(x)$ k־פעמים. עתה נבחין בעיקרון הליבה של האלגוריתם: x_k היא סדרה המוגדרת מעל שדה גלואה $\mathbb{F}_n$, לכן בשלב מסויים הערכים שיופיעו בשדה **יחלו לחזור על עצמם**. מכאן ש־x_k קשורה בקשר הדוק לסדרה אחרת – $\{x'_{k'}\, mod\, p\}$. וודאי, איננו יודעים מיהו p כך שלא ניתן לחשב את $\{x'_{k'}\, mod\, p\}$ במפורש באלגוריתם, ועדיין בסדרה זו טמון רעיון הליבה של האלגוריתם. אם נניח שאיברי הסדרות נבחרים באופן אקראי ובהתפלגות אחידה (דיסקרטית)[8], אזי לפי פרדוקס יום ההולדת[9] הסדרה $\{x'_{k'}\, mod\, p\}$ תופיע הרבה לפני $\{x_k\}$ עצמה.
ברגע שנמצא בסדרה איבר שחוזר על עצמו – נגיע למחזוריות של הסדרה, שהרי כל איבר תלוי רק בזה שקודם לו (ולכן נחזור לבצע מעגלים). [10]
אם כן. עבור כל שני אינדקסים בסדרה, i, j (כלומר הם מייצגים את איברי הסדרה x_i, x_j), הראשון יתקדם לאיבר הבא בסדרה, והשני לאיבר שלאחריו. עתה. נבדוק האם $gcd(x_i - x_j, n) \neq 1$. אם אכן התשובה לכך חיובית, זה אומר שמצאנו מחזוריות בתת סדרה $x_k\, mod\, p$, שהרי $x_i\, mod\, p = x_j\, mod\, p$. במילים אחרות, מאחר שההבדל בין כל שני מספרים שחלוקתם ב־p תוביל לשארית זהה, הוא כפל בסקלר כלשהוא, אז אם תוצאת ה־gcd שונה מ־1, $x_i\, mod\, p$ ו־$x_j\, mod\, p$ זהים עד כפל בסקלר – ובכך מצאנו את המספר הראשוני הנדרש. שיטה זו נקראית Floyd's cycle-finding algorithm.[11]
לסיום, נציין שלשיטה זו חיסרון ממשמעותי – בשלב מסויים gcd יתן תוצאה שונה מ־1 גם כשהיא לא משקפת את עיקרון החלוקה שלעיל. למשל, היא יכולה להחזיר את n עצמו כי שתי הסדרות עלולות להיות חופפות ולחזור על עצמן באותו הזמן. במקרים אלו – אלגוריתם ראו נכשל. אם כך, איך מתמודדים עם קושי זה? ניתן לנסות את האלגוריתם עם בחירת פולינום אחר (למשל $f(x) = x^2 + 3$) או להשתמש באלגוריתם ה־Trail Division שהוצגה לעיל כ־fallback.

---

[7] נשים לב שנוכל לבחור n טבעי כיוון שאם $n < 0 \Rightarrow -n > 0$ ולכן כל שנצטרך זה לכפול בהופכי של n. יתרה מכך, אם n זוגי, או שמדובר בחזקה של 2 – ואז זוהי הצגתו בפירוק לגורמים ראשוניים, או שקיים $m \in \mathbb{Z}$ כך ש־$n = 2 \cdot m - 1$ ולכן נוכל לקבל לאחר כמות מסויימת של פעולות חילוק ב־2 מספר אי זוגי.
[8] ראו: https://en.wikipedia.org/wiki/Discrete_uniform_distribution.
[9] מומלץ לקרוא על הפרדוקס: https://en.wikipedia.org/wiki/Birthday_paradox.
[10] אגב, מכאן נולד שמו של האלגוריתם – על שם האות היוונית ראו (ρ) שצורתה מזכירה את שיטת המחזוריות, כשהערכים $x_1\, mod\, n, \ldots, x_n\, mod\, n$ מוצגים בתור קדקודי גרף.
[11] ניתן לקרוא עליה כאן: https://en.wikipedia.org/wiki/Floyd%27s_cycle-finding_algorithm/

9.3 מימוש אלגוריתם rho של פולרד

משזאת נאמר, ניגש למימוש האלגוריתם. נבחר כפולינום את $f(x) = x^2 + 1$. בנוסף, נניח ש־$rand(a, b)$ פונקציה המגרילה מספר שלם בקטע $[a, b)$. אם כך, נגדיר את הפונקציה הבאה, שמקבלת n **מספר חיובי <u>ואי זוגי</u>:**

Algorithm 2 Pollard's Rho Algorithm

```
procedure pollardRho(n) :
  x ← rand(1, n − 1)
  y ← x
  p ← 1
  while p == 1 :
    x ← f(x)
    y ← (f ∘ f)(x)
    p ← gcd (|x − y| , n)
  if p == n then :
     return −1 # Failed to find p with the chosen polynomial
  return p
```

מימוש אלגוריתם זה יחזיר את p – אחד הגורמים הראשונים שמרכיבים את n, או מינוס 1 אם לא היה ניתן למצוא את p באמצעות בפולינום שנבחר. עתה, עליכם להשתמש בו עד לפירוק מלא של n למספרים ראשונים, כשעבור כל מספר אתם מבצעים את אלגוריתם הפירוק המלא (כלומר מנסים להפעיל את האלגוריתם של פולרד ואם הוא נכשל – עליכם להשתמש ב־Trail Division כדי לפתור את התרגיל),