מבני נתונים, סמסטר א' תשפ"א,
תרגיל תכנות 1

מועד הגשה: יום חמישי ה- 17.12.20 עד חצות. לא יינתנו הארכות!

הנחיות כלליות:

א. התרגיל הינו תרגיל חובה.

ב. התרגיל ניתן להגשה בבודדים או בזוגות, אך לא בקבוצות גדולות יותר.

ג. גם תלמידים החוזרים על הקורס מחויבים בהגשת התרגיל, ולא משנות הנסיבות.

ד. ציון "נכשל" בתרגיל, או הגשה מאוחרת שלו ללא אישור מתאים שניתן מראש על ידי המרצה, יגררו ציון "נכשל" סופי בקורס.

ה. איחור בהגשה יאושר רק במקרה של מילואים, מחלה או לידה, וגם זאת רק בתנאי שהפנייה למרצה בנושא נעשתה לפני מועד ההגשה המקורי של התרגיל.

מטרת התרגיל: כפל מספרים ארוכים על ידי שלושה אלגוריתמים שונים והשוואת יעילותם.

1. עליכם לכתוב תכנית ב- C++ המכילה שלוש פונקציות מרכזיות שמטרתן להכפיל מספרים ארוכים. הפונקציה הראשונה תכפיל את המספרים על ידי כפל ארוך בדומה לשיטה שנלמדת בבית ספר, ושתי הפונקציות האחרות יממשו את האלגוריתם של Karatsuba שיתואר בהמשך, פעם אחת בצורה רקורסיבית ופעם אחת ללא רקורסיה.
2. כמו-כן עליהם להגיש קובץ PDF נפרד שבו תנתחו ניתוח תיאורטי את זמן הריצה של שתיים מהפונקציות: הכפל הארוך והאלגוריתם הרקורסיבי של Karatsuba, כפונקציה של האורך n של שני המספרים שמכפילים.

תיאור הבעיה:

יהיו x,y שני מספרים טבעיים גדולים מאוד, כל אחד כולל n ספרות כאשר הספרות הן מהתחום {0,…,9}. שני המספרים נשמרים בשני מערכים הנקראים x, y כשכל מערך באורך n והספרה ה- i של כל מספר נמצאת בתא ה i של המערך.

כך למשל אם x = 3456 אז הוא ייוצג כך במערך

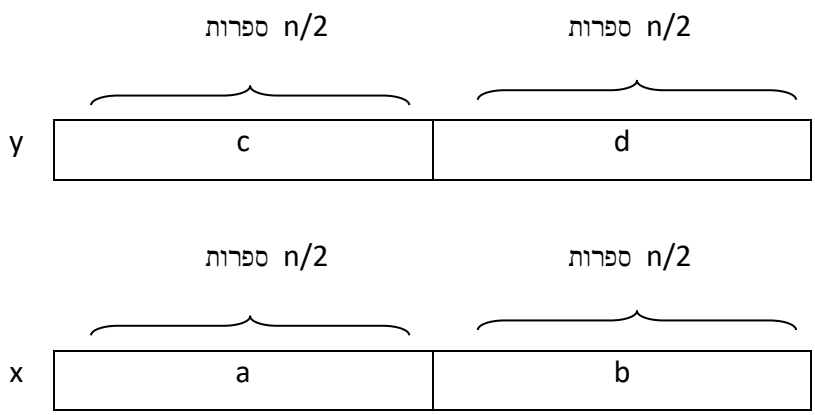
3	4	5	6

<u>האלגוריתם לכפל ארוך</u> של מספרים מוגדר כזכור כך:

			1	2	3	4
		×				
			2	4	5	9
		1	1	1	0	6
		6	1	7	0	
	4	9	3	6		
2	4	6	8			
3	0	3	4	4	0	6

<u>אלגוריתם Karatsuba</u> למכפלה של שני מספרים כאלו x ו-y מוגדר בעזרת הרעיון הרקורסיבי הבא:

נחלק את x ואת y לשני חלקים באורך n/2 כפי שאפשר לראות בציור הבא:



כאשר a מכיל את n/2 הספרות ה- most significant של x ו- b מכיל את n/2 הספרות ה- least significant של x (בדומה עם c ו-d במספר y).

נשים לב שמתקיים:

$$\boxed{x = a \cdot 10^{n/2} + b, \quad y = c \cdot 10^{n/2} + d}$$

האלגוריתם יחשב את המכפלה באופן רקורסיבי על פי הנוסחה הבאה:

$$\boxed{x \cdot y = (a \cdot c) \cdot 10^{n} + ((a+b) \cdot (c+d) - a \cdot c - b \cdot d) \cdot 10^{n/2} + (b \cdot d)}$$

חישבו היטב מהן הקריאות הרקורסיביות הנדרשות וכמה קריאות רקורסיביות שונות יש, והיכן אפשר להסתפק בפעולות מקומיות של האלגוריתם או לשמור תוצאות ביניים וכך לחסוך בקריאות רקורסיביות.

תיאור מפורט של התכנית:

1. קלט: התכנית תקבל כקלט בשלוש שורות נפרדות, שלושה מספרים שלמים ואי-שליליים n,x,y. המספר הראשון n הוא מספר הספרות של המספרים x ו- y. בכל אחד מהמספרים x,y יהיו בדיוק n ספרות, כולל אפסים מובילים אם יש צורך. כל המספרים ייכתבו משמאל לימין החל מהספרה המשמעותית ביותר ועד לספרה הכי פחות משמעותית (ספרת האחדות). כך למשל, קלט אפשרי תקין הוא:

   4

   0001

   3467

   שימו לב שבמספר n לא יהיו אפסים מובילים.

   אין להניח שהקלט תקין ולכן עליכן לבדוק כמובן את תקינות הקלט!

2. פלט: התכנית תדפיס את לכל היותר 2n הספרות של המכפלה x·y מהחשובה ביותר ועד להכי פחות חשובה שלוש פעמים, פעם עבור האלגוריתם לכפל מספרים ארוך, פעם עבור האלגוריתם הרקורסיבי ופעם עבור האלגוריתם שממיר רקורסיה במחסנית. **הפלט לא יכלול אפסים מובילים!**

   עבור כל אחת מהתשובות יירשם גם לקובץ טקסט בשם Measure.txt זמני הריצה של הפונקציות על פי ההנחיות בהמשך.

3. שימו לב: הקלט ייקרא מקלט סטנדרטי (מקלדת) והכתיבה היא לפלט סטנדרטי (מסך).
4. אלגוריתם לכפל ארוך: התכנית תכפיל את המספרים על ידי האלגוריתם לכפל ארוך.
5. אלגוריתם רקורסיבי: התכנית תפעיל פונקציה רקורסיבית לפי האלגוריתם של Kartsuba שתחשב את המכפלה x·y על ידי שימוש בנוסחה הרקורסיבית הנתונה למעלה. הפונקציה תקבל 2 מערכים בגודל n ואת n עצמו כמערכי קלט, ומערך שלישי בגודל 2n כמערך פלט עבור התוצאה.
6. אלגוריתם שמשתמש במחסנית: קראו בהמשך את "תיאור שיטת התרגום של אלגוריתם רקורסיבי לאיטרטיבי" וכתבו פונקציה **לא רקורסיבית** שעושה שימוש במחסנית ומחשבת את המכפלה x·y לפי האלגוריתם של Kartsuba אבל ללא הרקורסיה. הפונקציה תקבל את אותם פרמטרים כמו הפונקציה הרקורסיבית.

מבני הנתונים:

לצורך כתיבת התכנית עליכם לממש (בין היתר) את המחלקות הבאות:

1. מחסנית Stack – מחסנית של איברים מטיפוס ItemType שתכיל את הנתונים הדרושים לצורך ביטול הרקורסיה.
2. איבר מסוג ItemType – האיברים שיישמרו במחסנית.

אסור להשתמש במחלקות stack, list של STL אך מותר להשתמש ב- vector ו-string.

מחסנית Stack

עליכם **לממש** (אסור להשתמש ב- STL בשום צורה שהיא) מחסנית של נתונים **מסוג ItemType**. המחסנית תמומש כ**רשימה מקושרת חד-כיוונית**. אין להניח הנחות על מספר האיברים המקסימלי שיישמרו במחסנית במהלך התכנית.

הקפידו לכתוב את הפונקציות הדרושות לביצוע הפעולות הבסיסיות המוגדרות על מחסנית:

- IsEmpty()
- MakeEmpty()
- void Push(ItemType item)
- ItemType Pop(void)

איבר במחסנית ItemType: הגדירו מחלקה (נתונים ו- methods) שתהיה הטיפוס של איברי המחסנית.

מדידת זמני ריצה: עליכם להשתמש בקטע הקוד הבא שמודד את זמן הריצה של הפונקציה fun ורושם אותו לקובץ (כמובן תוך ביצוע ההתאמות הדרושות)

```
auto start = chrono::high_resolution_clock::now();

// unsync the I/O of C and C++.
ios_base::sync_with_stdio(false);

fun();// Here you put the name of the function you wish to measure

auto end = chrono::high_resolution_clock::now();

// Calculating total time taken by the program.
double time_taken =
       chrono::duration_cast<chrono::nanoseconds>(end - start).count();

time_taken *= 1e-9;

ofstream myfile("Measure.txt"); // The name of the file

myfile << "Time taken by function <name-of-fun> is : " << fixed
       << time_taken << setprecision(9);
myfile << " sec" << endl;
myfile.close();
```

השלבים לביצוע התרגיל

א. קראו תחילה היטב את ההנחיות של התרגיל והבינו את האלגוריתמים המתוארים בו. מומלץ להריץ את האלגוריתמים "הרצה יבשה" על דוגמאות.

ב. קראו היטב את עמודים 6-14 בהמשך התרגיל שמסבירים כיצד להפוך אלגוריתם רקורסיבי לאלגוריתם המבצע את אותה המשימה ללא רקורסיה על ידי שימוש במחסנית.

(שימו לב: הפתרונות המוצגים בחומר הקריאה נעזרים במחסנית. באופן דומה עליכם להשתמש במחסנית לצורך שינוי מימושו של האלגוריתם של Kartsuba לכזה שאינו רקורסיבי. פתרון שלא יעשה שימוש במחסנית ובטכניקה המוצעת, אלא יציע אלגוריתם איטרטיבי חלופי ייפסל על הסף).

ג. תכננו את ה-design של התוכנית שלכם: בחרו אילו מחלקות תממשו, החליטו על data members מתאימים ועל methods רלוונטיים לכל מחלקה.

ד. כתבו מימוש מלא לכל המחלקות שייעשה בהן שימוש במסגרת התוכנית.

ה. כתבו שלוש פונקציות: אחת לכפל מספרים ארוך, השנייה מממשת את אלגוריתם Karatsuba **הרקורסיבי** והשלישית מממשת את האלגוריתם של Kartsuba **ללא רקורסיה**.

ו. ממשו את התוכנית שלכם, המריצה את שלוש הפונקציות הנ"ל בזו אחר זו ומחשבת גם את זמני הריצה של שלוש הפונקציות.

ז. **שימו לב:** לא ניתן להניח דבר על הקלט או על חוקיותו. במקרה של שגיאה כלשהי בקלט עליכם לצאת מהתכנית ולהדפיס wrong input. יהיו טסטים במאמא שיכילו בין היתר קלטים שגויים, אם כי התכנית שלכם יכולה בהחלט להיבדק גם על קלטים שלא נמצאים במאמא.

ח. **ודאו שהתכנית שלכם מתקמפלת ורצה גם בויז'ואל סטודיו 2019 ולא רק במאמא.**

הנחיות הגשה:

יש להגיש במערכת mama במקום המיועד להגשה את הקבצים הבאים:

1. קובץ readme שהוא קובץ טקסט פשוט – לא word, שיכיל את כל פרטי ההגשה הבאים:
   כותרת – תרגיל תכנות מס' 1 במבני נתונים תשפ"א סמסטר א'.
   שורה מתחת - שמות המגישים, מספרי ת.ז. שלהם ומספר הקבוצה של כל אחד מהם (מותר להגיש עם בן זוג מקבוצה אחרת).
2. כל קבצי הקוד בסיומות cpp. ו- h.
3. קובץ PDF שמכיל ניתוח תיאורטי של זמני הריצה של האלגוריתם לכפל ארוך ושל הגרסה הרקורסיבית של האלגוריתם של Kartsuba כפונקציה של האורך n של המספרים שמכפילים.

שימו לב! הגשה שאינה בפורמט הנדרש תידחה אוטומטית.

תיאור שיטת התרגום של אלגוריתם רקורסיבי לאיטרטיבי

רקורסיה היא שיטה אלגנטית לפתרון בעיות רבות, אולם הקריאות הרקורסיביות דורשות לעיתים זמן חישוב ומשאבי זיכרון רבים. זאת מכיוון שכל קריאה רקורסיבית דורשת לשמור במחסנית זמן הריצה של התוכנית את כל המידע הדרוש: ערכם של המשתנים הלוקאליים, ערכם של הפרמטרים והכתובת של השורה שאליה צריך לחזור בסיום הרקורסיה. לכן במקרים רבים רצוי לקחת אלגוריתם רקורסיבי ולהפכו לאלגוריתם שאינו רקורסיבי.

ביטול רקורסיית זנב

רקורסית זנב Tail Recursion: אלגוריתם (פונקציה) משתמש ברקורסית זנב אם הקריאה הרקורסיבית היא הפעולה האחרונה שמתבצעת באלגוריתם. במקרים אלה קל למדי להפוך את הרקורסיה ללולאת while. נתחיל בדוגמה פשוטה:

```
void CountDown(int n)
{
        if (n == 0)
                cout << n;
        else {
                cout << n;
                CountDown(n–1);
        }
}
```

והנה הגרסה הלא רקורסיבית:

```
void CountDown(int n)
{
        while (n > 0){
                cout << n;
                n = n–1;
        }
        if (n == 0)
                cout << n;
}
```

מה עשינו כאן? במקום תנאי העצירה של הרקורסיה השתמשנו בלולאת while שנמשכת כל עוד לא הגענו לתנאי העצירה המקורי של הרקורסיה. כמו-כן, במקום לשלוח את הערך n–1 לקריאה הרקורסיבית, שמרנו אותו במשתנה (הפרמטר) הלוקאלי n עצמו. שימו לב גם שתנאי העצירה בפונקציה הרקורסיבית מתבצע בסוף שרשרת הקריאות הרקורסיביות. לכן בגרסה האיטרטיבית הזזנו אותו אחרי לולאת ה-while שמיישמת את הקריאות הרקורסיביות.

ועוד הסבר מדוע זה עובד כאן כל כך בקלות: כפי שאמרנו למעלה כאשר יש קריאה רקורסיבית, המחשב שומר במחסנית זמן הריצה את המשתנים הלוקאליים ואת מספר השורה שיש לחזור אליה בפונקציה הקוראת. כאשר חוזרים מהקריאה הרקורסיבית נשלפים ערכים אלה ממחסנית זמן הריצה. אולם, כשמדובר ברקורסית זנב, הרי הקריאה הרקורסיבית היא הפעולה האחרונה שמתבצעת בפונקציה הקוראת. לכן, אחסון המשתנים במחסנית זמן הריצה הוא צעד מיותר לחלוטין במקרה זה! כל מה שנחוץ כדי להיפטר מהרקורסיה הוא לעדכן במשתנים לוקאליים את הפרמטרים שהיינו מעבירים לקריאה הרקורסיבית כך שיקבלו את הערך החדש שלהם, ולחזור לתחילת האלגוריתם. את זה אפשר להשיג כמובן על ידי לולאה (או צעד goto בסגנון תכנותי פחות יפה).

הנה דוגמה נוספת ל- Tail Recursion:

```
void strange(int m, int n)
{
        if ( m > 0){
                cout << "This is strange\n";
                strange(n–1, m–1);
        }
}
```

שוב הקריאה הרקורסיבית היא הדבר האחרון שנעשה בפונקציה. שימו לב שבקריאה הרקורסיבית מחליפים n ו- m תפקידים. והנה הגרסה הלא רקורסיבית:

```
void strange(int m, int n)
{
        int  temp1, temp2;

        while (m > 0){
                cout << "This is stange\n";
                temp1 = n–1; temp2 = m–1;
                m = temp1; n = temp2;
        }
}
```

השתמשנו בלולאת while שנמשכת כל עוד לא הגענו לתנאי העצירה המקורי של הרקורסיה. גם כאן, במקום לשלוח את הערכים n–1, m–1 לקריאה הרקורסיבית, שמרנו אותם במשתנים n, m בעצמם (תוך שימוש במשתני העזר temp1, temp2 כדי להחליף את התפקידים של n,m).

לפני שנעבור לדון ברקורסיה שאיננה רקורסית זנב, הנה דוגמה נוספת. להלן תוכנית רקורסיבית לפתרון בעיית מגדלי הנוי.

```
void Hanoi(int n, int from, int to, int using)
{
        if (n == 1)
                cout<< "Move disk 1 from" << from << "to" << to << "\n" ;
        else if (n > 1){
                Hanoi( n–1, from, using, to);
                cout << "Move disk" << n << "from" << from << "to" << to << "\n";
                Hanoi( n–1, using, to, from);
        }
}
```

שימו לב שיש כאן שתי קריאות רקורסיביות. אנחנו נבטל רק את הקריאה הרקורסיבית האחרונה:

```
void Hanoi(int n, int from, int to, int using)
{
        while (n > 1) {
                Hanoi( n–1, from, using, to);
                cout << "Move disk" << n << "from" << from << "to" << to << "\n";
                n = n – 1;   Swap(from,  using);
        }
        if (n == 1)
                cout<< "Move disk 1 from" << from << "to" << to << "\n" ;
}
```

לו היינו משאירים את משפט ה- if הראשון בתחילת הפונקציה (כלומר לפני לולאת ה- while), אז היינו מסיימים את הפונקציה בלי לטפל במקרה הבסיסי של n = 1. לכן משפט ה- if הוזז לסוף הפונקציה.
כאמור, הקריאה הרקורסיבית הראשונה בפונקציה הזאת איננה רקורסית זנב (כי יש אחריה עוד הרבה פקודות לביצוע), ולכן היא נשארה בינתיים. בהמשך נראה מה אפשר לעשות במקרה זה.

רקורסית זנב עם ערך מוחזר:

לעתים הרקורסיה מחזירה ערך. שימו לב שברקורסית זנב הערך המוחזר בסופו של דבר מהרקורסיה כולה הוא פשוט הערך שמוחזר מהקריאה הרקורסיבית האחרונה, כלומר מתנאי העצירה (זאת מכיוון שברקורסית זנב כשחוזרים בחזרה מהקריאות הרקורסיביות לא עושים דבר נוסף, אלא רק ממשיכים להחזיר את הערך הזה). כך למשל הפונקציה הבאה שמקבלת מצביע לראש רשימה מקושרת ומחזירה את האיבר האחרון של הרשימה:

```
struct node{
        int data;
        struct node *next;
}

struct node * Last (struct node *list)
{
        if (list == NULL)
                return NULL;
        if (list–>next == NULL)
                return list;
        else
                return last (list–>next);
}
```

הפעולה האחרונה של הפונקציה לפני שהיא מחזירה ערך היא קריאה רקורסיבית ולכן זו רקורסית זנב. הנה הגרסה האיטרטיבית:

```
struct node * Last (struct node * list)
{
        if (list == NULL)
                return NULL;
        while (list–>next != NULL)
                list = list–>next;
        if (list–>next == NULL)
                return list;
}
```

ביטול רקורסיה כללית בעזרת מחסנית

כאשר יש יותר מקריאה רקורסיבית אחת, או כאשר הרקורסיה אינה בסוף התוכנית צריך לרוב להיעזר במחסנית כדי לשמור את הנתונים החיוניים. באופן עקרוני נשמור במחסנית את ערכם של המשתנים הלוקאליים, את ערכם של הפרמטרים שדרושים לרקורסיה ואת השורה שאליה צריך לחזור, באופן שמדמה את פעולת מחסנית זמן הריצה. אנחנו נראה שתי שיטות לעשות זאת, שתיהן מבוססות למעשה על העיקרון הזה של שימוש במחסנית.

נדגים את שתי השיטות על ידי שימוש בדוגמה של מגדלי הנוי. הנה שוב הפונקציה הרקורסיבית:

```
void Hanoi(int n, int from, int to, int using)
{
        if (n == 1)
                cout<< “Move disk 1 from” << from << “to” << to << “\n” ;
        else if (n > 1){
                Hanoi( n–1, from, using, to);
                cout << “Move disk” << n << “from” << from << “to” << to << “\n”;
                Hanoi( n–1, using, to, from);
        }
}
```

שיטה 1:

כאמור, נשתמש במחסנית כדי לשמור את הערך של המשתנים הלוקאליים, הפרמטרים שמועברים לרקורסיה, והשורה שאליה צריך לחזור:

המשתנים הלוקאליים והפרמטרים שמועברים לקריאה הרקורסיבית: במקום שבו יש קריאה רקורסיבית בפונקציה המקורית, נבצע שתי פעולות Push:

1. פעולת Push ראשונה תשמור לנו את המצב לפני הקריאה הרקורסיבית: כלומר נכניס למחסנית את ערכם של המשתנים הלוקאליים (בפונקציה Hanoi אין כאלה) ואת הערך הנוכחי של הפרמטרים n, from, to, using. קריאה זו נחוצה לנו כי כאשר נחזור "לפונקציה הקוראת" נצטרך לשחזר את מצב כל המשתנים והפרמטרים.
2. פעולת Push שנייה תשמור את ערכם של הפרמטרים שמועברים לקריאה הרקורסיבית: כך למשל, בדוגמה של Hanoi נכניס למחסנית את הערכים n–1, using, to, from שמועברים כפרמטרים לקריאה הרקורסיבית הראשונה.

מכיוון שמדובר במחסנית, הרי פעולת ה- Pop הבאה תוציא קודם את הערכים שנשמרו בפעולת ה- Push השנייה, ולכן אכן נבצע את "הקריאה הרקורסיבית" **לפני** שנחזור "לפונקציה הקוראת".

השורה שאליה צריך לחזור: כידוע, קריאה רקורסיבית גורמת לנו "לקפוץ" לתחילתה של הפונקציה. לעומת זאת, כשאנו חוזרים מהרקורסיה אנחנו חוזרים לשורה שמיד אחרי הקריאה. במקרה של הפונקציה Hanoi יש שלוש שורות שאליהן אנחנו יכולים להגיע כתוצאה מקריאה רקורסיבית או כשחוזרים מהרקורסיה:

1. השורה START שהיא פשוט תחילת הפונקציה,
2. השורה AFTER_FIRST שהיא השורה מיד אחרי הקריאה הרקורסיבית הראשונה,
3. השורה AFTER_SECOND שהיא השורה מיד אחרי הקריאה הרקורסיבית השנייה (במקרה של הפונקציה Hanoi, לא מתבצעות למעשה שום פקודות בשורה הזאת, ולכן זה מיותר כאן. אולם אנו מסבירים כאן את העיקרון הכללי).

לכן, אנחנו נגדיר שלושה קבועים כנ"ל ונכניס למחסנית בכל אחת מפעולות ה- Push שתוארו למעלה, גם את אחד מהקבועים START, AFTER_FIRST, AFTER_SECOND. כשנבצע פעולת Pop יישלפו, יחד עם הערכים המעודכנים של המשתנים והפרמטרים, גם השורה line שאליה צריך לחזור.

המבנה הרקורסיבי יוחלף לכן במבנה של משפטי if-else (ואפשר גם משפט switch), ולפי הערך של line נדע איזה משפט תנאי אמור להתבצע.

כיצד מאתחלים את הכול: מיד בתחילת הפונקציה הלא רקורסיבית, נכניס למחסנית את ערכם הנוכחי של המשתנים והפרמטרים, ואת הקבוע START שמסמן שרק התחלנו את הרקורסיה. כעת נשתמש במבנה כללי של לולאת while שתתבצע כל עוד המחסנית אינה ריקה. שימו לב שכל פעם שמסתיימת איטרציה של לולאת ה- while, זה שקול לסיומה של קריאה רקורסיבית בפונקציה המקורית, ואז אנחנו שולפים את "הקריאה הקודמת" מראש המחסנית וממשיכים בביצוע הפונקציה.

הערה:

כדי לפשט את החיים, נגדיר רשומה מסוג Item, ונכניס למחסנית איברים מסוג זה. ברשומה יהיו שדות עבור כל אחד מהפרמטרים והמשתנים הלוקאליים של הפונקציה, וכן שדה עבור המשתנה line שהוספנו כדי לזכור באיזו שורה אנחנו. הנה הרשומה המתקבלת עבור מגדלי Hanoi:

```
struct Item{            // an item that will be stored in Stack which simulates the recursion.
        int n;
        int from;  int to;  int using;
        int line;
};
```

והנה סוף סוף הפונקציה המתאימה למגדלי הנוי, והפעם ללא רקורסיה בכלל:

```
void Hanoi(int n, int from, int to, int using)
{
        Stack S;      // Stack which simulates the recursion.
        Item Curr;   // Values of current  "recursive call"
        Item Next;   // Values of next "recursive call".

       Curr = (n, from, to, using, START);
       S.Push( Curr );
       while ( !S.IsEmpty( ) ){
              Curr = S.Pop( );
               if ( Curr.line == START ){
                      if (Curr.n == 1)
                           cout<<"Move disk 1 from"<<Curr.from<<"to"<<Curr.to<<"\n" ;
                      else{
                          Curr.line = AFTER_FIRST;
                           S.Push( Curr );
                           Next =  (Curr.n – 1, Curr.from, Curr.using, Curr.to, START);
                           S.Push( Next );
                       }
              }
              else if ( Curr.line == AFTER_FIRST){
                     cout<<"Move disk"<<Curr.n<<"from"<<Curr.from<<"to"<<Curr.to<<"\n";
                     Curr.line = AFTER_SECOND;
                      S.Push( Curr );
                      Next = (Curr.n – 1, Curr.using,  Curr.to, Curr.from, START);
                      S.Push( Next );
              {
              else if ( Curr.line == AFTER_SECOND){
                    // In this case do nothing.
              }
     }
}
```

<u>שיטה 2:</u>

נשים לב שבשיטה שראינו זה עתה, כאשר יש "קריאה רקורסיבית" באיטרציה כלשהי של לולאת ה-while, אז אנחנו מבצעים שתי פעולות Push למחסנית. במקרה כזה, באיטרציה הבאה של הלולאה נבצע פעולת Pop ואז יישלפו מהמחסנית הערכים שנשמרו בפעולת ה- Push השנייה של האיטרציה הקודמת. לכן למעשה, אפשר לחסוך את פעולת ה- Push השנייה. מתקבל המבנה הכללי הבא שדומה מאוד למבנה הקודם, אבל חוסך פעולת Push אחת. אנחנו ניעזר במבנה האיטרטיבי של לולאת ה- while כדי להעביר את הערכים המעודכנים של המשתנים והפרמטרים "לקריאה הרקורסיבית" הבאה. הפעם במקום שני המשתנים Curr, Next שהיו לנו בשיטה הקודמת, יהיה לנו רק משתנה אחד Curr שישמור את הערכים הנכונים עבור "הקריאה הנוכחית".

<u>המשתנים הלוקאליים והפרמטרים שמועברים לקריאה הרקורסיבית:</u> במקום שבו יש קריאה רקורסיבית בפונקציה המקורית, נבצע את שתי הפעולות הבאות:

1. פעולת Push שתשמור לנו את המצב לפני הקריאה הרקורסיבית: כלומר נכניס למחסנית את ערכם של המשתנים הלוקאליים ואת הערך הנוכחי של הפרמטרים. כמו קודם, קריאה זו נחוצה לנו כי כאשר נחזור "לפונקציה הקוראת" נצטרך לשחזר את מצב כל המשתנים והפרמטרים.
2. אחר כך נעדכן את המשתנים הלוקאליים והפרמטרים כך שיכילו את הערך הנכון עבור "הקריאה הרקורסיבית" החדשה.

<u>השורה שאליה צריך לחזור:</u> גם כאן יהיו לנו קבועים START, AFTER_FIRST, AFTER_SECOND שיסמנו לנו לאיזו שורה line צריך ללכת באיטרציה הבאה של הלולאה. בפעולת ה- Push שתוארה למעלה, נכניס למחסנית גם את אחד מהקבועים הנ"ל. גם כאן, כשנבצע פעולת Pop יישלפו, יחד עם הערכים המעודכנים של המשתנים והפרמטרים, גם השורה line שאליה צריך לחזור.

<u>כיצד מאתחלים את הכול:</u> עלינו לדעת בכל איטרציה של לולאת ה- while האם אנחנו באיטרציה שבה אמורים לשלוף מהמחסנית ערך על ידי Pop (כלומר האם חוזרים ל"קריאה רקורסיבית קודמת" ואז מבצעים Pop, או שמבצעים כרגע "קריאה רקורסיבית" חדשה ואז אין צורך ב- Pop). לכן, נגדיר משתנה נוסף returnFromRecursion שערכו יהיה true אם אנחנו חוזרים כעת "לקריאה" קודמת, וערכו יהיה false אם התחלנו עכשיו "קריאה רקורסיבית" חדשה. בהתחלה ערכו יהיה כמובן false. כעת נשתמש במבנה כללי של לולאת do-while שתתבצע כל עוד המחסנית אינה ריקה. בכל פעם שמסתיימת איטרציה של לולאת ה- while, נבדוק אם returnFromRecursion = true, ואם כן נשלוף מראש המחסנית את "הקריאה הקודמת" ונמשיך בביצוע הפונקציה.

נקבל את הפונקציה האיטרטיבית הבאה:

```
void Hanoi(int n, int from, int to, int using)
{
        Stack S;      // Stack which simulates the recursion.
        Item Curr;   // Values of current  "recursive call"
        int returnFromRecursion = 0;

       Curr = (n, from, to, using, START);

       do{
            if (returnFromRecursion)
                    Curr = S.Pop( );
            if ( Curr.line == START ){
                    if (Curr.n == 1){
                          cout<< "Move disk 1 from" << Curr.from << "to" << Curr.to << "\n" ;
                           returnFromRecursion = 1;
                    else{
                          Curr.line = AFTER_FIRST;
                           S.Push( Curr );
                           Curr.n = Curr.n–1;  Swap(Curr.using, Curr.to);
                           Curr.line = START;
                           returnFromRecursion = 0;
                    }
            }
            else if ( Curr.line == AFTER_FIRST){
                  cout << "Move disk" << n << "from" << Curr.from << "to" << Curr.to << "\n";
                  Curr.line = AFTER_SECOND;
                   S.Push( Curr );
                  Curr.n = Curr.n–1;  Swap(Curr.from, Curr.using);
                  Curr.line = START;
                  returnFromRecursion = 0;
            }
            else if (Curr.line == AFTER_SECOND){
                    returnFromRecursion = 1;
            }
       }
       while ( !S.IsEmpty( ) );
}
```

רקורסיה שמחזירה ערך:

גם במקרה הכללי של מספר כלשהו של קריאות רקורסיביות, ייתכן כמובן שהרקורסיה מחזירה ערך. נחזור לדוגמה של מגדלי הנוי, הפעם כאשר היא מחזירה ערך. הערך המוחזר על ידי הפונקציה בדוגמה הוא מספר הקריאות הרקורסיביות שהיא מבצעת, או לחלופין מספר הצעדים הנדרשים לפתרון בעיית מגדלי הנוי. הנה הפונקציה הרקורסיבית המעודכנת שמחזירה ערך.

```
int Hanoi(int n, int from, int to, int using)
    {
            int count1, count2;

             if (n == 1){
                        cout<< "Move disk 1 from" << from << "to" << to << "\n" ;
                        return 1;
             }
             else if (n > 1){
                        count1 = Hanoi( n–1, from, using, to);
                        cout << "Move disk" << n << "from" << from << "to" << to << "\n";
                        count2 = Hanoi( n–1, using, to, from);
                        return count1 + count2 + 1;
             }
    }
```

כדי להיפטר מהרקורסיה, שוב נעזרים במחסנית, אולם הפעם שומרים בה גם את ערכם של המשתנים הלוקאליים count1, count2. לכן הרשומה הדרושה לנו במקרה זה היא מהצורה:

```
struct Item{            // an item that will be stored in Stack.
        int n;
        int from;  int to;  int using;
        int count1;  int count2;
        int line;
};
```

בנוסף יהיה לנו משתנה returnValue שישמור את הערך המוחזר עד כה, ובסיום הפונקציה פשוט נחזיר אותו.

הנה הפונקציה ללא רקורסיה בשיטה ה- 1:

```
int Hanoi(int n, int from, int to, int using)
{
        Stack S;
        Item Curr, Next;
        int returnValue;    // value returned by function.

       Curr = (n, from, to, using, 0, 0, START);
       S.Push( Curr );
       while ( !S.IsEmpty( ) ){
              Curr = S.Pop( );
               if ( Curr.line == START ){
                      if (Curr.n == 1){
                           cout<<"Move disk 1 from"<<Curr.from<<"to"<<Curr.to<<"\n" ;
                           returnValue = 1;
                      }
                      else{
                          Curr.line = AFTER_FIRST;
                           S.Push( Curr );
                          Next =  (Curr.n – 1, Curr.from, Curr.using, Curr.to, 0, 0, START);
                           S.Push( Next );
                      }
              }
              else if ( Curr.line == AFTER_FIRST){
                      Curr.count1 = returnValue;
                     cout<<"Move disk"<<Curr.n<<"from"<<Curr.from<<"to"<<Curr.to<<"\n";
                      Curr.line = AFTER_SECOND;
                      S.Push( Curr );
                      Next = (Curr.n – 1, Curr.using,  Curr.to, Curr.from, 0, 0, START);
                      S.Push( Next );
              }
              else if ( Curr.line == AFTER_SECOND){
                     Curr.count2 = returnValue;
                     returnValue = Curr.count1 + Curr.count2 + 1;
              }
      }
      return returnValue;
}
```

והנה הפונקציה בשיטה ה- 2:

```
void Hanoi(int n, int from, int to, int using)
    {
        Stack S;
        int returnFromRecursion = 0;
        Item Curr;
        int returnValue; ;    // value returned by function.

        Curr = (n, from, to, using, START);

         do{
            if (returnFromRecursion)
                    Curr  = S.Pop( );
            if (Curr.line == START ){
                    if (Curr.n == 1){
                        cout<< "Move disk 1 from" << Curr.from << "to" << Curr.to << "\n" ;
                        returnFromRecursion = 1;
                        returnValue = 1;
                    else{
                        Curr.line = AFTER_FIRST;
                        S.Push(Curr );
                        Curr.n = Curr.n–1;  Swap(Curr.using, Curr.to);
                        Curr.line = START;
                        returnFromRecursion = 0;
                    }
            }
            else if ( Curr.line == AFTER_FIRST){
                cout << "Move disk" << n << "from" << Curr.from << "to" << Curr.to << "\n";
                Curr.count1 = returnValue;
                Curr.line = AFTER_SECOND;
                S.Push(Curr );
                Curr.n = Curr.n–1;   Swap(Curr.from, Curr.using);
                Curr.line = START;
                returnFromRecursion = 0;
            }
            else if (Curr.line == AFTER_SECOND){
                returnFromRecursion = 1;
                Curr.count2 = ReturnValue;
                returnValue = Curr.count1 + Curr.count2 + 1;
            }
        }
        while ( !S.IsEmpty( ) );
        return returnValue;
}
```