חומר טכני לראיון עבודה

1. הסבר כללי:
   - חלק "צפוי"
   - חלק "בלתי צפוי"
   - דוגמא לתיאור של פרוייקט

2. רשימת שאלות:
   - אינטל
   - רפא"ל
   - חברה שאני לא זוכר את השם שלה
   - בי.איי.טי.אם
   - איי.בי.אם
   - אלביט
   - מייקרוסופט
   - קיי.אל.איי
   - צורן
   - ספקטרום דיינאמיקס
   - פריסקייל
   - צ'ק פוינט
   - די.אס.פי.ג'י
   - ראד לייב
   - נייס
   - סאמסונג
   - ג'אנגו
   - דיסקרטיקס
   - ג'י.אי
   - אם-סיסטמס
   - אופטיבייס
   - קוואלקום
   - קוריג'נט
   - דיון נטוורקס
   - סיווה די.אס.פי
   - שאלות ששמעתי מאחרים

3. חומר כללי:
   - סמלים בתוכנה
   - פונקציות מחלקה
   - תכנות מונחה עצמים
   - ניהול שכבות הזיכרון
   - ניהול תהליכים וחוטים

4. תשובות לחלק מהשאלות...

השלב הטכני בראיונות עבודה מתחלק לשני חלקים:

חלק "צפוי"

- בחלק הזה אתה מתבקש לתאר פרוייקט כלשהו שעשית. סביר להניח שאם אתה סטודנט ללא ניסיון עבודה, אז זה יהיה פרוייקט מהלימודים, ואם יש לך ניסיון קודם בעבודה, אז זה יהיה פרוייקט שעבדת עליו.
- אפשר להתכונן אל החלק הזה מראש. ככל שתתכונן אליו יותר טוב, כך תוכל לעשות רושם יותר טוב, ובנוסף גם תעביר בו יותר זמן ותשאיר פחות זמן לחלק השני, שאליו אי אפשר כל כך להתכונן מראש.
- נקודה חשובה נוספת: ייתכן שגם תישאל לגבי דברים מסוימים שציינת בקורות החיים שלך (דברים שאתה יודע או שיש לך ניסיון בהם). לכן, במידה ויש כאלה, כדאי להתכונן ולחזור גם עליהם.

חלק "בלתי צפוי"

- בחלק הזה שואלים אותך שאלות טכניות או חידות. השאלות הטכניות יכולות להיות דברים ספציפיים שקשורים לתוכנה או לחומרה, החל מה-Low Level (המעבד או מערכת ההפעלה) ועד ל-High Level (שפת תכנות כלשהי). לחלופין, הן יכולות להיות בעיות שונות שאתה מתבקש למצוא להן פתרון אלגוריתמי. החידות הן בדרך כלל שאלות הגיון, שצריך להמיר לבעיות ואז לפתור באופן דומה (פתרון אלגוריתמי).
- אי אפשר ממש לצפות מראש איזה שאלות תישאל בחלק הזה. בדרך כלל המראיין לא מצפה שתחשוב לבד ותשלוף ישר את הפתרון הסופי, אלא שתתאר בקול רם איך אתה חושב לפתור את הבעיה. זאת אומרת, מצפים יותר לראות את צורת החשיבה שלך לכיוון הפתרון. הרבה פעמים גם מכוונים אותך תוך כדי.
- מהניסיון שלי, לרוב הבעיות יש פתרון פשוט ובנאלי, שהחסרון שלו מתבטא בכך שהוא איטי או דורש הרבה זיכרון (סיבוכיות זמן או מקום גבוהה). כדאי בדרך כלל להציג את הפתרון הזה בתור התחלה (אם מזהים אחד כזה, כמובן) ולהדגיש שזה הפתרון המיידי שעולה לראש. אחרי זה אפשר לציין את החסרונות שלו, ואז לנסות למצוא (ביחד עם המראיין) שיפורים.

דוגמא לתיאור של פרוייקט

1. תיאור כללי: ...
2. מעבד: ...
3. מערכת הפעלה: ...
4. סביבת פיתוח: ...
5. שפת תכנות: ...
6. בקרת תצורה: ...
7. תיאור מפורט: ...
8. חבילת עבודה עיקרית: ...

ריכזתי כמה שאלות טכניות מתוך כל מיני ראיונות עבודה. לא את כל השאלות זכרתי, כי בחלק מהמקרים רשמתי אותן די הרבה זמן אחרי הראיון עצמו. מעבר לזה, השתדלתי לא לרשום יותר מפעם אחת שאלות שחזרו על עצמן כמה פעמים. בקיצור, לא כדאי להסיק מכמות השאלות או מהתוכן שלהן לגבי הקושי של הראיון בכל אחת מהחברות...

אינטל (חיפה)

1. יש לך מערך עם כדורים אדומים, צהובים וירוקים. אתה צריך לסדר אותו כך שכל הכדורים האדומים יהיו בהתחלה וכל הכדורים הירוקים יהיו בסוף. אין לך שטח זיכרון נוסף להשתמש בו.
2. יש לך רשימה מקושרת. כתוב פונקציה שהופכת אותה.

רפא"ל (לשם)

3. נתון אולם. אנשים נכנסים אליו ויוצאים ממנו דרך שער, אשר מאפשר מעבר של בן אדם אחד בכל רגע נתון. ברשותך גלאי שמזהה מעבר של בן אדם. תכנן מערכת שתדע כמה אנשים נמצאים בתוך האולם בכל רגע נתון. לצורך העניין, מותר לך להשתמש בגלאי אחד או יותר. האם יכולה להיווצר בעיה במערכת שתיכננת?
4. נתונים Task'ים שקיימים במערכת ההפעלה. כל Task מתחיל בנקודת זמן מסוימת, מסתיים בנקודת זמן מסוימת, ובמהלך ה"חיים" שלו עובר ממצב Running למצב Pending (ולהיפך). הזמן שהוא נמצא במצב Running כל פעם הוא קבוע, והזמן שהוא נמצא במצב Pending כל פעם הוא גם קבוע (אבל הם לא בהכרח זהים). עבור כל Task קיים אובייקט שמתאר אותו (זמן ההתחלה שלו, זמן הסיום שלו, משך הזמן שהוא נמצא במצב Running כל פעם, ומשך הזמן שהוא נמצא במצב Pending כל פעם). הוסף למחלקה פונקציה שמקבלת את הזמן הנוכחי במערכת ומחזירה את מצב ה-Task שהאובייקט מתאר (Running, Pending, Not-Alive).
5. הרחב את המחלקה כך שתוכל לדעת כמה אובייקטים קיימים במערכת בכל רגע נתון.
6. מה זה static ב-C? מה זה static ב-++C?

חברה שאני לא זוכר את השם שלה (תל אביב)

7. יש לך רשימה של N שירים. אתה רוצה להשמיע את כולם, כל יום בסדר אחר. כתוב פונקציה שתסדר אותם כל יום בסדר אקראי. אתה יכול להיעזר בפונקציה Rand, שמחזירה מספר אקראי.
8. אתה כותב מחלקה, שאתה רוצה שאחרים יגזרו (ירשו) ממנה מחלקות לשימושם הפרטי. אתה רוצה שאף אחד לא יוכל ליצור עצם (Instance) של מחלקת הבסיס שאתה כותב (אלא רק עצמים של מחלקות שנגזרות ממנה). איך תממש את המחלקה כך שלא יהיה ניתן ליצור Instance'ים שלה?

בי.איי.טי.אם (יוקנעם)

9. נתונות 100 מנורות בטור. כולן כבויות. איש ראשון עובר ולוחץ על כל מפסק. איש שני עובר ולוחץ על כל מפסק שני. איש שלישי עובר ולוחץ על כל מפסק שלישי. ככה עוברים 100 אנשים. איזה מנורות יישארו דלוקות בסוף?
10. משחק – שני אנשים יושבים אחד מול השני, כשבאמצע שולחן עגול. לכל אחד יש מספר בלתי מוגבל של מטבעות עגולים. כל אחד בתורו מניח מטבע על השולחן. מותר להניח איפה שרוצים, אבל לא על מטבעות אחרים שכבר נמצאים על השולחן. השחקן הראשון שלא יכול יותר לשים מטבעות על השולחן, מפסיד. תאר שיטה לניצחון בטוח.
11. יש לך שני סלים, עשרה כדורים אדומים ועשרה כדורים שחורים. הכדורים מפוזרים בין שני הסלים. אני צריך להוציא כדור אדום בניסיון הראשון (בלי להסתכל). איך היית מחלק את הכדורים כך שיהיה לי הכי הרבה סיכוי להצליח?
12. יש לך שני כדורי זכוכית ובניין בן 100 קומות. ישנה קומה מסוימת, שממנה והלאה הכדורים יישברו אם תזרוק אותם (החל מהקומה הזאת ומעלה). אתה צריך לגלות איזה קומה זאת (תזכור – יש לך רק שני כדורים "לבזבז"). המטרה שלך היא לעשות את זה במספר קטן ככל האפשר של ניסיונות.

איי.בי.אם (חיפה)

13. אתה נוסע על כביש חד-סטרי. כל כמה זמן אתה עובר מול בית. יש לך מצלמה שיכולה לשמור תמונה אחת. בסוף הכביש נמצא שומר. אתה צריך להביא לו תמונה של בית אקראי. אתה לא יודע כמה בתים יש בינך לבין השומר. אתה יכול להשתמש במצלמה כמה פעמים שאתה רוצה, אבל רק התמונה האחרונה שתצלם בכל פעם היא זאת שתישאר בזיכרון. תאר שיטה להגיע לשומר, כשבמצלמה נמצאת תמונה של בית אקראי. לרשותך פונקציה Rand. בנוסף, עליך להוכיח שעבור N בתים, ההסתברות להגיע לשומר עם תמונה של כל אחד ואחד מהם היא זהה.
14. יש לך מערך של N מספרים. כתוב פונקציה שתסדר אותם בסדר אקראי. אתה יכול להיעזר בפונקציה Rand, שמחזירה מספר אקראי (ת'כלס, בדיוק כמו שאלה 7).

אלביט (חיפה)

לא ממש זוכר. הייתה בעיקר שאלה אחת, ארוכה מדי בשביל לתאר פה...

מייקרוסופט (חיפה)

15. מה זה פונקציות וירטואליות? איך התוכנית יודעת להגיע לפונקציה הנכונה כשפונקציה וירטואלית נקראת?
16. איך מממשים פונקציות עם מספר לא קבוע של ארגומנטים ב-C (כמו printf, למשל)?
17. שני חוטים רצים במקביל. בכל אחד ישנה לולאה, ושניהם צריכים להמתין אחד לשני במקום מסוים בלולאה (כל אחד בלולאה שלו) ורק אז להמשיך. פתור את הבעיה בעזרת אמצעי סינכרון שמוכרים לך, ובלי להשתמש בחוט נוסף. לשם הפשטות, אפשר להניח שהלולאות של שני החוטים הן זהות, ושנקודות ההמתנה שלהם בלולאות הן גם זהות. פתור את אותה בעיה עבור N חוטים.
18. כתוב פונקציה אשר מקבלת מחרוזת, ומדפיסה את כל הפרמוטציות האפשריות שלה.
19. כתוב פונקציה אשר מקבלת מחרוזת String ואוסף של תווים Set, ומחזירה את האינדקס של המופע הראשון של תו כלשהו מ-Set ב-String. הסיבוכיות שלה צריכה להיות סכום אורכי הקלט. ניתן להניח שהתווים הם בקוד ASCII. איזה בעיות יהיו בפונקציה אם התווים הם בקוד רחב יותר, ואיך ניתן לפתור אותן?
20. נתון מילון. איך תייצג אותו בצורה יעילה? תאר אלגוריתם שמקבל כקלט מילה ומוצא את כל המילים במילון שהן פרמוטציה כלשהי של מילת הקלט.
21. רוצים לתכנן מעבד עם פקודות באורך קבוע. כל פקודה מורכבת מפעולה + אופרנדים, באורך כולל של 12 ביטים. האופרנדים האפשריים הם כתובות באורך של 3 ביטים. סט הפקודות הנדרש כולל:
    - 4 פעולות שמקבלות שלושה אופרנדים כל אחת.
    - 255 פעולות שמקבלות אופרנד אחד כל אחת.
    - 16 פעולות שלא מקבלות אופרנדים כלל.

    א. האם ניתן לתכנן מעבד כזה? הסבר.

    ב. נסח תנאי שיבדוק אם ניתן לתכנן מעבד עבור המקרה הכללי:
    - K_1 פעולות שמקבלות P_1 אופרנדים באורך של L_1 ביטים.
    - K_2 פעולות שמקבלות P_2 אופרנדים באורך של L_2 ביטים.
    - ...
    - K_n פעולות שמקבלות P_n אופרנדים באורך של L_n ביטים.
22. נתון עץ בינארי (לכל צומת 2 בנים לכל היותר). העץ לא מאוזן ולא ממוין, וייתכנו רשומות זהות בצמתים שונים. רוצים להעביר את העץ ממחשב אחד למחשב אחר. תכנן אלגוריתם שיקודד את העץ לקובץ, ואלגוריתם שיפענח את הקובץ חזרה לעץ.

קיי.אל.איי (מגדל העמק)

23. מה זאת מחלקה אבסטרקטית?
24. איזה הבדלים קיימים בין פונקציה וירטואלית לפונקציה וירטואלית טהורה?
25. מה זה Copy Constructor ולמה חשוב להגדיר אותו?
26. למה משמשת ה-V-Table?
27. מה זה Singleton ואיך היית ממש את זה?
28. מה ההבדל בין סמאפור ל-Mutex?
29. איך מתבצעת תקשורת בין תהליכים ואיך מתבצעת תקשורת בין חוטים?

צורן (חיפה)

30. ממש ב-C פונקציה שמקבלת שתי מחרוזות ומשווה ביניהן.
31. הגדר ב-C ממשק של פונקציות עבור תור (FIFO).
32. נתון חניון עם קיבולת של M מכוניות. לפני שער הכניסה G1 יש חיישן S1, שעולה כאשר מגיעה מכונית אל השער, ויורד לאחר שהיא נכנסת לחניון. לפני שער היציאה G2 יש חיישן S2, שעולה כאשר מגיעה מכונית אל השער, ויורד לאחר שהיא יוצאת מהחניון. כתוב תוכנית שמנהלת את החניון. לרשותך פונקציה Open(i) שפותחת את שער מספר i, פונקציה Close(i) שסוגרת את שער מספר i, ופונקציה Test(i) שבודקת את הסטאטוס של חיישן מספר i.
33. מה ההבדל בין סמאפור בינארי ל-Mutex?
34. מה ההבדל בין Process ל-Thread?
35. נתון מערך של מיליון תווים. בהינתן תו כלשהו, צריך לבדוק אם הוא נמצא במערך.

    א. כתוב פונקציה שמקבלת תו ובודקת אם הוא נמצא במערך.

    ב. מצא דרך לשפר את זמן הריצה של הפונקציה שכתבת.

ספקטרום דיינאמיקס (טירת הכרמל)

36. ישנו חוואי בעל פרה אחת. כל בוקר החוואי קם, הולך לנהר, לוקח מים והולך להשקות את הפרה. מצא את הדרך הקצרה ביותר שבה החוואי יכול לבצע את המשימה. ניתן להניח שהנהר הוא פס ישר ברוחב קבוע כלשהו, ושהחוואי והפרה נמצאים באותו צד שלו.
37. נתונה צורה כלשהי במישור (כאוסף של קודקודים במערכת הצירים XY). בהינתן נקודה, איך תוכל לדעת אם היא נמצאת בתוך הצורה, מחוץ לצורה או על השפה של הצורה?

פריסקייל (הרצליה פיתוח)

38. נתון מערך גדול מאד שמאותחל בהצהרה: int array[] = {…}. קיימת בתוכנית פונקציה אחת שמשתמשת במערך לצורך קריאה בלבד. אפשר להגדיר את המערך כמשתנה לוקאלי בתוך הפונקציה או כמשתנה גלובאלי מחוץ לפונקציה. איך יושפעו גודל התוכנית (קובץ ה-Image) ומהירות הביצוע שלה בכל אחד מהמקרים?
39. בנה מערך עם המספרים 0 עד N-1, מסודרים בצורה אקראית. לרשותך פונקציה Rand, שמחזירה מספר אקראי.
40. נתונה טבלה בגודל N × N, ובכל משבצת רשום ערך כלשהו. מתחילים ממשבצת כלשהי בשורה הראשונה, ובכל שלב מותר לעבור משבצת אחת למטה או משבצת אחת למטה וימינה או משבצת אחת למטה ושמאלה. המטרה היא לאסוף סכום ערכים גדול ככל האפשר. מצא דרך לחשב את הסכום המקסימלי האפשרי.

צ'ק פוינט (רמת גן)

41. נתונים שני מפסקים ונורה. לכל מפסק ישנם שני מצבים (מורם או מורד), ולנורה ישנם שני מצבים (דלוקה או כבויה). כל שינוי במצב של אחד המפסקים, גורם לשינוי במצב של הנורה. תכנן מעגל חשמלי עבור המערכת הזאת. פתור את אותה בעיה עבור שלושה מפסקים.
42. נתונה רשימה של זמרים. בנוסף, נתון יחס סדר חלקי ביניהם (כלומר, עבור חלק מהזמרים, ידוע מי יותר טוב ממי). בהינתן שני זמרים, עליך לקבוע מי יותר טוב מביניהם, או להכריז שלא ניתן לדעת.
43. יש לך מעבד עם זיכרון של 1MB. על המעבד רצות אפליקציות שונות, שעשויות לבצע הקצאה ושחרור של זיכרון דינאמי. מערכת ההפעלה שרצה על המעבד מספקת את הממשק הבא:
    - void* malloc() – מקצה שטח זיכרון של 8 בתים, ומחזירה את הכתובת שבה הוא מתחיל.
    - void free(void* p) – מקבלת כתובת שבה מתחיל שטח זיכרון של 8 בתים, ומשחררת אותו.

    תכנן שיטה לנהל את הזיכרון, וממש את הממשק כך שיאפשר הקצאה של 8 בתים כל עוד יש מקום פנוי.

די.אס.פי.ג'י (הרצליה פיתוח)

44. האם תוכנית שנכתבה ב-C או ב-C++ מתחילה לרוץ מהפונקציה main או מבצעת משהו לפני כן?
45. אתה מפתח תוכנה שאמורה לרוץ על מעבדים שונים, ולכן גם נבנית באמצעות קומפיילרים שונים. לאיזה נקודות (הבדלים אפשריים בין המעבדים או הקומפיילרים) חשוב להתייחס במהלך כתיבת הקוד, על מנת שהתוכנה תעבוד בצורה תקינה?
46. הפונקציה memcopy מעתיקה נתונים ממקום אחד בזיכרון למקום אחר בזיכרון.
    א. הגדר וממש את הפונקציה. האם ניתן לשפר את זמן הריצה שלה?
    ב. האם ייתכן מצב מסוים, שבו הפונקציה לא תעתיק את הנתונים במדויק?
47. מתי Context Switch בין Task'ים יכול להתבצע, ומתי הוא לא יכול להתבצע?
48. נתונה תוכנית שכוללת כמה Task'ים. בנוסף, מוגדר בה משתנה גלובאלי a. בחלק מה-Task'ים מתבצעת קריאה לפונקציה כלשהי, שבה מוגדר משתנה לוקאלי b.
    א. נניח שבפונקציה רשום a=b. האם במהלך ריצת התוכנית תמיד תוכל לדעת בוודאות מהו הערך של a? אם לא, איך תשנה את הקוד על מנת להבטיח את זה?
    ב. נניח שבפונקציה רשום b=a. האם במהלך ריצת התוכנית תמיד תוכל לדעת בוודאות מהו הערך של b? אם לא, איך תשנה את הקוד על מנת להבטיח את זה?
49. נתונה הפונקציה הבאה:

```
char* strcat(char* first, char* last)
{
        char buff[100];
        sprintf(buff,"%s%s",first,last);
        return buff;
}
```

מה לא תקין בפונקציה?

50. איך תוכנית יודעת לחזור משגרת פסיקה (ISR) לנקודה שבה היא הייתה לפני כן?

ראד לייב (תל אביב)

51. יש לך רשימה מקושרת חד-כיוונית מעגלית. כל צומת בה מכיל רשומת נתונים (data) ומצביע לצומת הבא (next).
בהינתן צומת ששייך לרשימה, עליך לנתק אותו ממנה ולמחוק אותו מהזיכרון. סיבוכיות הזמן המותרת היא O(1).
52. יש לך רשימה מקושרת חד-כיוונית בגודל N. עליך לבדוק אם יש בה מעגל. סיבוכיות הזמן המותרת היא O(N).

נייס (רעננה)

שאלו רק לגבי פרוייקטים שעבדתי עליהם...

סאמסונג (הרצליה פיתוח)

ראיון של משאבי אנוש. כשהיא שאלה מה הממליצים שרשמתי יגידו עליי, אמרתי לה שאני לא יודע, ושאולי יהיה יותר פשוט אם היא תתקשר אליהם במקום לשאול אותי. כנראה שבגלל זה לא עברתי...

ג'אנגו (נתניה)

53. C – נתונות פונקציות להקצאה ולשחרור של זיכרון דינאמי:
- OS_small_malloc(int sz) – יכולה להקצות 512 בתים לכל היותר.
- OS_large_malloc(int sz) – יכולה להקצות 512 בתים לכל הפחות.
- OS_free(void* p, int sz) – משחררת sz בתים מכתובת p בזיכרון.

א. ממש את הפונקציה void* malloc(int sz). תפקידה הוא להקצות זיכרון בכל גודל, ובכפולות של 16 בתים.
ב. ממש את הפונקציה void free(void* p). תפקידה הוא לשחרר זיכרון שהוקצה באמצעות הפונקציה הקודמת.
ג. שכלל את הפונקציה free, כך שתדפיס הודעת שגיאה כאשר מתבצע ניסיון לשחרר זיכרון שכבר שוחרר בעבר.

54. C – נתונה פונקציה להשהיה: Sleep(int seconds). ממש את הממשק הבא, ע"מ לאפשר שימוש ב-Timer'ים:
- הפונקציה void CreateTimer(void (*func)(), int seconds), שתפקידה הוא ליצור Timer.
- הפונקציה void StartTimers(), שתפקידה הוא להפעיל את כל ה-Timer'ים שקיימים במערכת.

כל Timer שמופעל, צריך לבצע השהיה של מספר שניות ולאחר מכן לקרוא לפונקצית ה-Callback.
דוגמא לשימוש ב-Timer'ים:

```
CreateTimer(func1,3);
CreateTimer(func2,7);
CreateTimer(func3,10);
StartTimers();
```

הפעולות הבאות צריכות להתבצע (בצורה טורית):
- השהיה של 3 שניות ואחריה קריאה ל-func1.
- השהיה של 4 שניות ואחריה קריאה ל-func2.
- השהיה של 3 שניות ואחריה קריאה ל-func3.

דיסקרטיקס (נתניה)

55. נתונה הפונקציה הבאה:

```
void func(int* x)
{
	x = (int*)malloc(sizeof(int));
	*x = 17;
}
```

מה תדפיס התוכנית הבאה, ומה לא תקין בה?

```
void main()
{
	int y = 42;
	func(&y);
	printf("%d",y);
}
```

ג'י.אי (טירת הכרמל)

56. נתונה התוכנית הבאה:

```
void main()
{
	const char* arr = "abcde";
	char* ptr = (char*)malloc(strlen(arr)+1);
	for (int i=0; i<strlen(arr)+1; i++)
		*ptr++ = arr[i];
	free(ptr);
}
```

מה לא תקין בתוכנית?

57. נתונים שני Thread'ים. כמו כן, נתונה הפונקציה הבאה:

```
int func()
{
	int a = 1;
	a++;
	return a;
}
```

ה-Thread הראשון מתחיל לרוץ וקורא לפונקציה. כשהוא מסיים את השורה הראשונה, מתבצע Context Switch.
ה-Thread השני מתחיל לרוץ וקורא לפונקציה. כשהוא מסיים את השורה השניה, מתבצע שוב Context Switch.
מהו הערך של המשתנה a בשלב הזה?

58. יש לך ארגז ובו 15 זוגות של גרביים (כלומר, 30 גרביים). כל זוג הוא בצבע שונה, וכל הגרביים מופרדות. מהי ההסתברות להוציא זוג גרביים (כלומר, שתי גרביים מאותו צבע) בניסיון הראשון?

אם-סיסטמס (רעננה)

59. איך ניתן לדעת מהי "צריכת ה-Stack" המקסימלית של תוכנית כלשהי?
60. תאר אלגוריתם לניהול מעלית.

אופטיבייס (הרצליה פיתוח)

61. מה קורה כאשר פונקצית מחלקה כלשהי מבצעת delete this?
62. ב-C++ אפשר להעביר משתנה By Reference. מהם היתרונות שיש לשיטה הזאת על פני שיטות אחרות?
63. יש לך מחלקה A, מחלקות B ו-C שיורשות ממנה, ומחלקה D שיורשת מהן. כמה אובייקטים מסוג A יש "בתוך" אובייקט מסוג D, ואיך ניתן לגשת לכל אחד מהם?

קוואלקום (חיפה)

64. כתוב פונקציה שמקבלת מטריצה ריבועית של מספרים (מספר שווה של שורות ועמודות) ואת גודלה של המטריצה, והופכת אותה (כלומר, מבצעת עליה Transpose) ללא שימוש בזיכרון נוסף.
65. תחנה A מעבירה Msg'ים לתחנה H בקצב של Msg/MilliSec. תחנה H מחזירה Ack'ים לתחנה A באותו קצב. לכל Msg יש מספר מזהה ייחודי. כל Ack שתחנה H מחזירה מתאים ל-Msg שתחנה A שולחת, אבל הסדר שבו ה-Ack'ים מוחזרים לא בהכרח זהה לסדר שבו ה-Msg'ים נשלחים. עבור כל Msg שתחנה A שולחת, תחנה H צריכה להחזיר Ack תוך שניה אחת לכל היותר. בין תחנה A לתחנה H, ישנה תחנה B, שתפקידה לזהות תקלות. תאר את מבנה הנתונים שיש לנהל בתחנה B, ואת אופן השימוש בו.

קוריג'נט (תל אביב)

66. נתונים שלושה Task'ים שרצים במערכת: A רץ בעדיפות גבוהה, B רץ בעדיפות בינונית, C רץ בעדיפות נמוכה. A ו-C חולקים משאב משותף כלשהו, וידוע שהם מסונכרנים ביניהם (כלומר, לפני ש-A ניגש אל המשאב המשותף, הוא ממתין ש-C יסיים להשתמש בו, ולהיפך). בשלב כלשהו, C ניגש אל המשאב המשותף, ולפני שהוא מסיים להשתמש בו, B מתחיל לרוץ. מה הבעיה במצב המתואר, ואיך ניתן לפתור אותה?

דיון נטוורקס (יקום)

67. יש לך רגיסטר של 32 ביטים. כתוב פונקציה שמוצאת את המיקום של הביט השלישי שערכו 1. איך תוכל לשפר את זמן הריצה של הפונקציה שכתבת?
68. נתונה מערכת מרובת Task'ים. לאחר שהיא מסיימת לעבוד, נוצר Log שמכיל את זמני ההתחלה וזמני הסיום של כל ה-Task'ים שרצו בתוכה. מצא את נקודת הזמן שבה רצו הכי הרבה Task'ים במקביל.

סיווה די.אס.פי (הרצליה פיתוח)

69. נתון כרטיס שכולל מעבד, זיכרון פנימי של 8KB וזיכרון חיצוני של 60KB. כמו כן, מוגדרות שלוש פעולות:
    1. Load(loc,buf) – טעינת 1KB מכתובת loc בזיכרון החיצוני לכתובת buf בזיכרון הפנימי (250 Cycles).
    2. Store(loc,buf) – טעינת 1KB מכתובת buf בזיכרון הפנימי לכתובת loc בזיכרון החיצוני (250 Cycles).
    3. Process(buf) – עיבוד של 1KB נתונים בכתובת buf בזיכרון הפנימי (600 Cycles).

    פעולות 1 ו-2 הן פעולות DMA, וניתן לבצע כל אחת מהן במקביל לפעולה 3 (אבל לא במקביל לפעולה השניה). כתוב תוכנית מהירה ככל האפשר, שקוראת 60KB של נתונים מהזיכרון החיצוני, מעבדת אותם בזיכרון הפנימי וכותבת אותם בחזרה לזיכרון החיצוני (מותר להניח שקוד התוכנית לא תופס מקום בזיכרון הפנימי).
70. נתונה הפונקציה הבאה:

```
long func(short x)
{
        long res = 0;
        static short array[20000];
        extern short const_array[20000];

        for (int i=19999; i>0; i--)
                array[i] = array[i-1];
        array[0] = x;

        for (int j=0; j<20000; j++)
                res += (long)array[j]*const_array[j];
        return res;
}
```

פעולת גישה למערך לוקחת חמישה Cycles'ים, וכל פעולה אחרת (+, -, *, = וכו') לוקחת Cycle אחד.

א. מה הפונקציה עושה, ומהו זמן הריצה שלה?

ב. נתונות שתי פעולות חדשות:
    1. inc_mod(i,n) ← i = (i+1)%n (הגדל את i ב-1 מודולו n).
    2. dec_mod(i,n) ← i = (i-1+n)%n (הקטן את i ב-1 מודולו n).

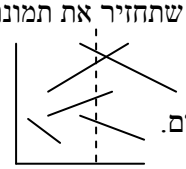
    כל אחת מהן לוקחת Cycle אחד. מצא דרך לשפר את זמן הריצה של הפונקציה, וחשב אותו מחדש.

שאלות ששמעתי מאחרים

71. אתה צריך לשלוח מכתב לחבר שלך, ואסור שאף אחד יוכל לקרוא את המכתב בדרך. לצורך כך, יש לשניכם מספר בלתי מוגבל של מזוודות ומנעולים קפיציים (מנעול קפיצי ניתן לסגור בלי מפתח). מצא פתרון לבעיה. אי אפשר, למשל, לשלוח מפתח בדואר, כי מישהו יכול לשכפל אותו בדרך ולהשתמש בו אחרי זה.
72. נסעת לכיוון אחד במהירות של 40 קמ"ש. באיזה מהירות אתה צריך לחזור, על מנת שהמהירות הממוצעת שלך (עבור כל הדרך) תהיה 80 קמ"ש?
73. נתונים שני משתנים. איך תחליף בין הערכים שלהם בלי להשתמש במשתנה נוסף?
74. נתונה טבלה של הרבה משתנים ויחסי גודל ביניהם (למשל: a<b, x>y וכו'). תאר אלגוריתם שיאפשר לך לסדר אותם בסדר עולה.
75. נתון כלא ובו N אסירים. בחצר הכלא יש מנורה שניתן להדליק ולכבות. מנהל הכלא מכנס את האסירים ומציע להם שחרור מהכלא, אם יעמדו במשימה הבאה: המנהל יוציא כל פעם אסיר כלשהו לחצר ואחר כך יחזיר אותו לתא. כאשר האסירים בטוחים שכל אחד מהם כבר יצא לחצר לפחות פעם אחת, הם צריכים להודיע על כך למנהל. בתחילת המשימה המנהל מאפשר לאסירים לצאת לחצר בשביל לתכנן ביחד דרך פעולה. במהלך המשימה אין לאסירים שום דרך לתקשר ביניהם (למעט המנורה כמובן). מצא שיטה שבעזרתה יוכלו האסירים להצליח במשימה.
76. נתון מערך בגודל n. כל המספרים בתחום [m,...,m+n+1] נמצאים במערך הזה, פרט לשניים, שהם לא המינימום או המקסימום. למשל, אם n=6, m=2, והמערך מכיל את המספרים 2,4,6,7,8,9, אז חסרים המספרים 3,5. המערך לא ממוין. כיצד ניתן לגלות בשני מעברים על המערך מי הם המספרים החסרים? אסור להשתמש בכמות זיכרון שתלויה בגודלו של המערך (כלומר, מותר להשתמש רק בכמות זיכרון קבועה).
77. נתון שולחן עגול מסתובב עם 4 מתגי לחיצה (לא ניתן לדעת אם מתג נמצא ב-On או ב-Off). באמצע השולחן יש נורה, אשר המתח עבורה מחובר למפסק ראשי נפרד. בהתחלה הנורה כבויה. ידוע שהנורה הזאת נדלקת רק כאשר כל המתגים נמצאים ב-On או כאשר כל המתגים נמצאים ב-Off. בכל צעד המפסק הראשי יורד, ואתה יכול ללחוץ על כל מתג שאתה רוצה (אחד או יותר). לאחר סיום הפעולה, מרימים את המפסק הראשי לראות אם הפעולה הצליחה (כלומר, הנורה נדלקה). אם לא, מורידים את המפסק הראשי, מסובבים את השולחן באופן אקראי ונותנים לך הזדמנות נוספת. הכול סימטרי, כך שאתה לא יודע איזה מתגים שינית בצעד הקודם. המטרה: הגדר אלגוריתם דטרמיניסטי, אשר בסיומו הנורה נדלקת.
78. תאר אלגוריתם לפתרון של סודוקו. כלומר, בהינתן טבלה של 3 × 3 ריבועים, כשכל ריבוע הוא 3 × 3 משבצות, מצא אלגוריתם שמחשב סידור של הספרות 1-9 בכל המשבצות, כך שכל ספרה מופיעה בדיוק פעם אחת בכל ריבוע בטבלה, בכל שורה בטבלה ובכל טור בטבלה.
79. נתון ביטוי שמכיל שלושה סוגים של סוגריים: ( ), [ ], { }.
    א. תאר אלגוריתם שבודק אם הביטוי הוא חוקי או לא.
    ב. אותה בעיה, כאשר קיים סוג רביעי של סוגריים, שבו אין הבדל בין סוגר שמאלי לסוגר ימני: | |.
80. נתון שטח בגודל 2^N × 2^N ללא המשבצת הימנית התחתונה. הוכח שניתן לרצף אותו בעזרת קבוצות של משבצות מהצורה  בלבד (מותר לסובב אותה).
81. כתוב תוכנית שבודקת כמה ביטים דלוקים יש בבית אחד. כעת כתוב אותה כך שתתבצע בזמן קבוע. מהו הטרייד אוף?
82. נתון בניין בן 100 קומות. ספיידרמן נמצא בקומת הקרקע ורוצה לטפס על הבניין. ספיידרמן מטפס ע"י קפיצות, כאשר הוא מסוגל לעלות קומה אחת או שתי קומות בכל קפיצה.
    א. כתוב תוכנית שמחשבת את מספר האפשרויות השונות (קומבינציות שונות של קפיצות), שבהן ספיידרמן יכול לטפס על הבניין.
    ב. מהי סיבוכיות הזמן שלה ומהי סיבוכיות הזיכרון שלה?
    ג. הראה שניתן לשנות את התוכנית כך שהיא תתבצע בזמן לינארי.
    ד. הראה שניתן לשנות את התוכנית כך שהיא תתבצע בזיכרון קבוע.
83. כתוב אוטומט סופי דטרמיניסטי, שמקבל כקלט מספר בינארי מהספרה הגדולה ביותר לספרה הקטנה ביותר, ובודק אם המספר מתחלק ב-5.
84. כתוב תוכנית שמחפשת "לולאה" ברשימה מקושרת חד-כיוונית, ומשתמשת בזיכרון בגודל קבוע.
85. נתון המשחק הבא: המראיין כותב שורה של מספרים כלשהם, כשהאילוץ היחידי הוא מספר זוגי של איברים (מספרים). לאחר מכן, כל אחד בוחר את האיבר הימני ביותר או השמאלי ביותר ובכך "לוקח" אליו את המספר (ומוציא אותו מהמשחק). כך, עד ש"לוקחים" את כל המספרים – פעם אתה, פעם הוא. השחקן שלוקח את סכום האיברים הגדול ביותר, מנצח. מצא את הטריק שמבטיח ניצחון למי שמקבל את התור הראשון.

שאלות ששמעתי מאחרים (המשך)

86. בהינתן מספר, איך אפשר לדעת אם הוא חזקה שלמה של 2, תוך שימוש ב-O(1) פעולות חישוב בלבד?
87. איך פונקצית מחלקה יודעת איזה אובייקט קרא לה? איך נבנה קוד האסמבלי של קריאה לפונקציה כזאת ע"מ שהיא תתבצע עם האובייקט הנכון?
88. נתונה תמונה בגודל M על N. כל פיקסל מיוצג באמצעות ביט בודד. כתוב פונקציה שתחזיר את תמונת הראי שלה.
89. נתון אוסף של קטעים במערכת הצירים XY.
כל קטע נתון כשתי נקודות במישור: $[(x_1,y_1) , (x_2,y_2)]$.
מצא את המספר הגדול ביותר של נקודות חיתוך בין ישר מאונך כלשהו לבין הקטעים.
לדוגמא, בשרטוט המצורף ישנן 4 נקודות חיתוך כאלה לכל היותר.



90. למה Constructor לא יכול להיות וירטואלי, ולמה Destructor צריך להיות וירטואלי?
91. פעולת הקצאה דינאמית מתבצעת בעזרת מערכת ההפעלה. כיצד ניתן למנוע את השימוש במערכת ההפעלה, כאשר מבצעים הקצאה דינאמית של אובייקט כלשהו ב-C++ (למשל, A* a = new A)?
92. תאר שיטה ליישם ב-C++ את מנגנון הפונקציות הווירטואליות ללא שימוש במילה השמורה virtual.
93. נתון מערך של N מספרים. מצא את המינימום והמקסימום, תוך שימוש ב-1½N פעולות השוואה לכל היותר.
94. נתונה תמונה בגודל M על N. כל פיקסל מיוצג באמצעות בית אחד. תאר אלגוריתם למציאת הריבוע המקסימלי של פיקסלים מאותו צבע.
95. נתונה מערכת הפעלה שבה הפונקציה malloc עובדת כרגיל, אבל הפונקציה free מקבלת, בנוסף למצביע לבלוק זיכרון, גם את גודל הבלוק. על המערכת הנ"ל רצה תוכנית שמקצה ומשחררת זיכרון באופן דינאמי. ממש פונקציות להקצאה ולשחרור של זיכרון, שבהן התוכנית תוכל להשתמש באופן המקובל בשפת C:
   א. void* MyMalloc(int size)
   ב. void MyFree(void* ptr)
96. נתונה תוכנית Real-Time שמשתמשת במאגר קבוע של N חוצצים (Buffer'ים). התוכנית נעזרת בשתי פונקציות: GetBuffer() – מחזירה חוצץ פנוי, FreeBuffer(buffer) – מקבלת חוצץ ומשחררת אותו אם הוא תפוס. הגדר מבנה נתונים שיאפשר ביצוע מהיר של שתי הפונקציות הנ"ל. אתחול המבנה יכול להתבצע בכל סיבוכיות.
97. נתון מעבד ובו 8 רגיסטרים של 4 ביט כל אחד. ישנן ארבע פעולות שניתן לבצע על רגיסטר Rx כלשהו:
   - INC Rx – הגדלת הערך שב-Rx ב-1.
   - DEC Rx – הקטנת הערך שב-Rx ב-1.
   - CLR Rx – איפוס הערך שב-Rx.
   - JUMP Rx LABEL – ביצוע קפיצה ל-LABEL, אם הערך שב-Rx שונה מאפס.

   א. בצע בעזרת הפעולות הנ"ל R3←R2*R1. האם ניתן לוותר על חלק מהפעולות ולממש אותן באמצעות הפעולות האחרות?
   ב. האם הקוד שכתבת עובד נכון גם עבור מספרים שליליים (כלומר, כאשר מתייחסים לערכי הרגיסטרים בשיטת המשלים ל-2)?
98. נתון מעבד שמחובר לאזור זיכרון שנקרא Prog ולאזור זיכרון שנקרא Data. ב-Prog נמצא קוד וב-Data נמצאים נתונים. החיבור ל-Data הוא באמצעות קו שעוביו 8 סיביות, וייתכן שחלקן מנותקות (מחזירות 0 קבוע). כתוב פונקציה שתיצרב ב-Prog, ותמצא את הסיבית המנותקת הראשונה מביניהן.
99. נתונים שני מעבדים עם גישה לזיכרון משותף, שמכיל משתנה בשם x. במעבד הראשון קיים קטע קוד שמגדיל את x ב-1 (x++), ובמעבד השני קיים קטע קוד שמקטין את x ב-1 (x--). בכל הפעלה של המערכת, x מאותחל ל-0, קטע הקוד הראשון רץ m פעמים וקטע הקוד השני רץ n פעמים (במקביל).
   א. האם מובטח שהערך של x בסיום הריצה של שני קטעי הקוד יהיה אותו ערך בכל פעם?
   ב. אם לא, הוסף לאחד מקטעי הקוד (או לשניהם) רצף פעולות שיבטיח את זה. מה יהיה הערך של x בסיום?

שאלות ששמעתי מאחרים (המשך)

100. האם פונקצית מחלקה סטאטית יכולה להיות וירטואלית? מדוע?

101. ספינה צריכה להגיע לאי, כאשר בדרך ישנם הרבה קרחונים. איך ניתן לחשב את המסלול הקצר ביותר שהספינה יכולה לעשות בשביל להגיע לאי בלי לעבור דרך הקרחונים? מותר להניח שמיקום הספינה ומיקום האי נתונים כנקודות במישור, ושכל קרחון מיוצג כמצולע (אוסף של קודקודים) במישור.

102. נתונים שני מיכלים. במיכל הראשון יש ליטר חלב ובמיכל השני יש ליטר סירופ שוקולד. לוקחים כוס מהמיכל הראשון, מוזגים למיכל השני ומערבבים. לאחר מכן לוקחים כוס מהמיכל השני, מוזגים בחזרה למיכל הראשון ומערבבים שוב. מהו היחס בין ריכוז החלב במיכל הראשון לריכוז סירופ השוקולד במיכל השני?

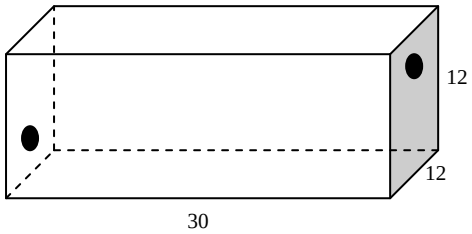
103. שלושה אנשים רוצים לדעת מהו ממוצע המשכורות שלהם. איך הם יכולים לחשב את הממוצע הזה בלי שאף אחד יגלה לאחרים מהי המשכורת שלו (ובלי להיעזר באיש נוסף)?

104. בהינתן מספר שבו ישנם N ביטים שערכם 1, מצא את הערך של N תוך שימוש ב-N איטרציות לכל היותר.

105. נתון מערך של N מספרים שלמים, שכולל את כל הערכים בתחום [1, N-1]. אחד המספרים מופיע במערך פעמיים. מצא את הערך שלו, תוך מעבר אחד על המערך ובלי להשתמש בזיכרון נוסף.

106. ממש הכפלה של מספר כלשהו ב-7, ללא שימוש בפעולת כפל או בפעולת חיבור.

107. נתון חדר שגודלו 12 × 12 × 30 מטר. הקיר השמאלי והקיר הימני הם בגודל 12 × 12 מטר. הקיר הקידמי, הקיר האחורי, הרצפה והתקרה הם בגודל 12 × 30 מטר. באמצע הקיר השמאלי ישנו שקע חשמל שנמצא מטר מעל הרצפה, ובאמצע הקיר הימני ישנו שקע חשמל שנמצא מטר מתחת לתקרה. ברור שניתן להעביר כבל באורך 42 מטר בין השקעים, אבל איך ניתן להעביר כבל באורך 40 מטר ביניהם?



108. שלושה גברים צריכים להיות עם אשה אחת, כאשר יש ברשותם רק שני קונדומים. איך ניתן לעשות את זה בצורה בטוחה עבור כולם (גם האשה וגם הגברים)?

109. נתון מספר שלם לא מסומן (unsigned int). הגדל אותו ב-1, ללא שימוש בפעולת חיבור או בפעולת חיסור.

110. נתונה סדרה של N מספרים. מצא את תת הסדרה הרצופה המקסימלית (מבחינת סכום איברים) שקיימת בה.

111. ידוע שאפשר למצוא איבר במערך ממוין תוך שימוש ב-O(log(N)) פעולות, ע"י חיפוש בינארי. נניח שסובבנו את המערך סביב ציר כלשהו, למשל: מ-{1,2,3,4,5} ל-{3,4,5,1,2}. מצא שיטה למצוא איבר במערך החדש (כאשר ציר הסיבוב לא ידוע לך), תוך שימוש באותה סיבוכיות זמן.

112. ממש שער XOR בעזרת שערי NAND בלבד.

113. אצן A ואצן B מתחרים בריצת 100 מטר. אצן A מסיים 5 מטרים לפני אצן B. בשביל לעשות את המירוץ יותר הוגן, הם מבצעים אותו שוב, אלא שהפעם אצן A מתחיל 5 מטרים מאחורי אצן B. בהנחה ששניהם רצים בדיוק באותו קצב שבו הם רצו בפעם הראשונה, מי מהם ינצח בפעם השניה? (בלי לבצע חישובים אלגבריים).

114. חולה מקבל מהרופא שלו בקבוק של 30 כדורים מסוג X ובקבוק של 30 כדורים מסוג Y. ההוראות של הרופא: "במשך 30 יום, אתה צריך לקחת כל יום כדור אחד מכל סוג, ואתה צריך לקחת את שני הכדורים ביחד". ביום הראשון, החולה מוציא כדור אחד מסוג X ושני כדורים מסוג Y (בטעות). כל הכדורים נראים בדיוק אותו דבר, כך שאין לו אפשרות לדעת איזה כדור הוא צריך להחזיר לבקבוק. הוא לא יכול לקנות בקבוקים חדשים, כי הם מאד יקרים. איך הוא יכול לקחת את התרופות לפי ההוראות שהרופא נתן לו?

שאלות ששמעתי מאחרים (המשך)

115. נתון מערך שמכיל N מספרים שלמים בתחום [0…N-1]. בדוק אם קיים בו ערך כלשהו שמופיע יותר מפעם אחת.

116. שני שחקנים משחקים זה מול זה. ישנה קופה משותפת עם 0 מטבעות, וכל שחקן בתורו שם בה 1, 2 או 3 מטבעות. בתחילת המשחק בוחרים מספר X כלשהו, והשחקן שמביא את הקופה ל-X מטבעות, מנצח. המגבלה היחידה על הערך של X, היא ש-X לא מתחלק בשלמות ב-4. מצא שיטה לניצחון בטוח עבור השחקן שמתחיל ראשון.

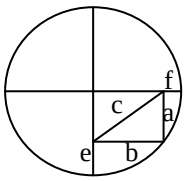
117. כמה פעמים ברציפות מופיעה הספרה 0 בסוף הייצוג העשרוני של המספר 100 עצרת?

118. C++ – איך ניתן לדעת מהו מספר האיברים:
   - א. במערך שהוקצה בצורה סטאטית (לדוגמא, Type Array[10])?
   - ב. במערך שהוקצה בצורה דינאמית (לדוגמא, Type Array=new Type[10])?

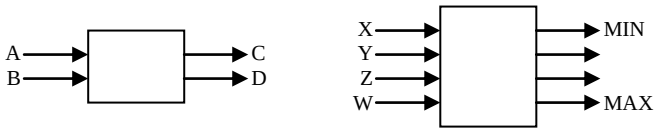
119. C++ – הסבר מה לא תקין בקטעי הקוד הבאים:
   - א. T* p=0; delete p;
   - ב. T* p=new T[10]; delete p;

120. C++ – מה זה RTTI ואיך זה עובד?

121. נתונים המעגל והמשתנים הבאים:
   - א. חשב את קוטר המעגל כפונקציה של המשתנים a, b, c, e, f.
   - ב. חשב את קוטר המעגל כפונקציה של המשתנים a, b, c בלבד.



122. נתונה יחידה לוגית בעלת שתי כניסות A ו-B ושתי יציאות C ו-D: C = min(A,B), D = max(A,B). באמצעות יחידות מסוג זה בלבד, עליך לבנות יחידה חדשה עם ארבע כניסות וארבע יציאות, כך שהמספרים ביציאות יהיו ממוינים בסדר עולה. המימוש צריך להיות פשוט ככל שניתן (שימוש במספר קטן ככל האפשר של יחידות).



123. נתון שעון בעל תדר כלשהו. באמצעות יחידות מסוג D-Flip-Flop ושערים לוגיים מכל סוג, צריך לבנות מחלק תדר לשעון (כלומר, שעון חדש שיעבוד בחצי מהתדר של השעון המקורי).

124. נתונים פתילים A ו-B. כאשר מדליקים את פתיל A בקצה, הוא מתחיל לבעור ומתכלה לאחר 60 דקות. כאשר מדליקים את פתיל B בקצה, הוא מתחיל לבעור ומתכלה לאחר 30 דקות. קצב הבעירה של שני הפתילים אינו קבוע (למשל, יכול להיות שכעבור 55 דקות יישרף רק חצי מפתיל A, וב-5 הדקות הבאות יישרף כל החצי השני שלו). כיצד ניתן למדוד 45 דקות בעזרת שני הפתילים? כעת נתונים פתילים A ו-C, כאשר פתיל C מתכלה גם הוא לאחר 60 דקות בקצב בעירה שאינו קבוע (ושאינו בהכרח זהה לקצב הבעירה של פתיל A). כיצד ניתן למדוד 45 דקות בעזרת שני הפתילים האלה?

125. הפיכת סדר הביטים ביחידת זיכרון:
   - א. כתוב פונקציה שמקבלת בית, ומחזירה את הבית ההפוך לו מבחינת סדר הביטים שלו.
   - ב. כתוב פונקציה שעושה את אותה פעולה עבור בית, בזמן קבוע ביחס למספר הביטים שלו.
   - ג. כתוב פונקציה שעושה את אותה פעולה עבור רגיסטר, בזמן קבוע ביחס למספר הביטים שלו.

סמלים בתוכנה:

סמלים בתוכנה נועדו לייצג ערכים כלשהם, ע"מ להקל על כתיבת הקוד ועל קריאתו.
כל הסמלים בתוכנה מוחלפים ע"י הקומפיילר בערכים שהם מייצגים:

- סמלי מאקרו, טיפוס ותבנית (#define, typedef, template) נודעו לייצג ערכים קבועים כלשהם.
  סמלים אלה מוחלפים כבר בשלב ה-Preprocessing, בערכים שנקבעו עבורם ע"י המתכנת.
- שאר הסמלים – שמות של משתנים ופונקציות – נועדו לייצג כתובות זיכרון במהלך ריצת התוכנית.
  סמלים אלה מוחלפים במהלך הקומפילציה עצמה, בכתובות שנקבעות עבורם ע"י הקומפיילר.

כשתוכנית נכנסת למצב ריצה, היא מקבלת מרחב זיכרון קבוע (Memory Address Space) שבו היא פועלת.
שמות של משתנים ופונקציות שהוחלפו במהלך הקומפילציה בכתובות, קיבלו בשלב הזה ערכים יחסיים ולא מוחלטים.
כלומר, כל הכתובות שאליהן התוכנית ניגשת, נמצאות למעשה ב-Offset כלשהו מתחילת מרחב הזיכרון שמוקצה עבורה.

מרחב הזיכרון שבו התוכנית פועלת, מורכב משלושה חלקים בסיסיים:

1. אזור נתונים (Data Section).
2. אזור קוד (Code Section).
3. מחסנית (Stack).

על מנת להריץ את התוכנית, המעבד משתמש ברגיסטרים באופן הבא:

- רגיסטר ה-IP (Instruction Pointer) מצביע תמיד על הפקודה הבאה לביצוע, שרשומה באזור הקוד.
- רגיסטר ה-SP (Stack Pointer) מצביע תמיד על המקום הבא שפנוי במחסנית.

כאמור, במהלך הקומפילציה, שמות של משתנים ופונקציות מוחלפים בכתובות זיכרון:

- שמות של משתנים מוחלפים בכתובות, אשר בזמן ריצה יהיו שייכות לאזור הנתונים או למחסנית.
- שמות של פונקציות מוחלפים בכתובות, אשר בזמן ריצה יהיו שייכות לאזור הקוד.

המשתנים שמשויכים לאזור הנתונים מתחלקים לארבעה סוגים:

1. משתנים גלובאליים לא סטאטיים – מוכרים בכל הקבצים בתוכנית.
2. משתנים גלובאליים סטאטיים – מוכרים רק בקובץ שבו הם מוגדרים.
3. משתנים לוקאליים סטאטיים – מוכרים רק בפונקציה שבה הם מוגדרים.
4. משתנים מחלקתיים סטאטיים – מוכרים רק במחלקה שבה הם מוגדרים (ובאובייקטים מסוגה).

ההבדל בין הסוגים השונים הוא אך ורק ב-Scope שבו הקומפיילר מכיר אותם. בזמן ריצה אין ביניהם שום הבדל.
בנוסף לזה, הכתובות של המשתנים האלה נשארות קבועות במשך כל ריצת התוכנית (ביחס לתחילת מרחב הזיכרון שלה).

המשתנים שמשויכים למחסנית הם המשתנים הלוקאליים שאינם סטאטיים.
השמות שלהם מוחלפים בכתובות, שבזמן ריצה יהיו בעצמן יחסיות לערך של רגיסטר ה-SP:

- עבור פונקציה כלשהי, מצב המחסנית (רגיסטר ה-SP) עשוי להיות שונה בכל פעם שהפונקציה נקראת.
- כלומר, מיקום המשתנים של הפונקציה ביחס לתחילת המחסנית עשוי להיות שונה בכל פעם שהפונקציה רצה.

לכן, הכתובות של המשתנים האלה לא בהכרח נשארות קבועות במשך כל ריצת התוכנית (אפילו בתוך מרחב הזיכרון שלה).

משתנים מחלקתיים שאינם סטאטיים, משויכים או לאזור הנתונים או למחסנית (בהתאם לאובייקט שאליו הם שייכים).

הפונקציות, שמשויכות כולן לאזור הקוד, מתחלקות לארבעה סוגים: {גלובאליות , מחלקתיות} × {סטאטיות , לא סטאטיות}.
גם כאן, בדומה למשתנים שמשויכים לאזור הנתונים, אין בין הסוגים השונים של הפונקציות שום הבדל בזמן ריצה.
הכתובות של הפונקציות נשארות, כמובן, קבועות במשך כל ריצת התוכנית (בלי קשר לסוגים שלהן).

לסיכום:

- כל הסמלים בתוכנית מייצגים ערכים כלשהם, שנקבעים עוד לפני ריצת התוכנית.
- שמות של משתנים ופונקציות מוחלפים במהלך הקומפילציה, ובמקומם באות כתובות זיכרון.
- כתובות אלה הן לא קבועות במרחב הזיכרון הפיזי של המחשב, אלא יחסיות לתחילת מרחב הזיכרון שמוקצה לתוכנית בכל פעם שהיא רצה. חלקן קבועות במרחב הזיכרון הזה וחלקן משתנות אפילו ביחס לתחילתו (במהלך ריצת התוכנית).

פונקציות מחלקה:

ההבדל בין פונקצית מחלקה לא סטאטית לפונקצית מחלקה סטאטית:

פונקציה לא סטאטית	פונקציה סטאטית
יכולה להיקרא רק ע"י אובייקט מסוג המחלקה שאליה היא שייכת	יכולה להיקרא ע"י המחלקה עצמה (וגם ע"י אובייקט, אבל אין לתוכן האובייקט שום השפעה עליה)
מקבלת כפרמטר גם את כתובת האובייקט שבאמצעותו היא נקראת (למרות שזה מוסתר מהמתכנת)*	מקבלת כפרמטרים רק את מה שמוגדר בהצהרת הפונקציה
בעלת גישה לכל השדות והפונקציות של המחלקה (סטאטיים ולא סטאטיים)	בעלת גישה רק לשדות הסטאטיים ולפונקציות הסטאטיות של המחלקה
יכולה להיות וירטואלית	לא יכולה להיות וירטואלית

*כתובת האובייקט יכולה להישלח דרך המחסנית או ברגיסטר כלשהו (תלוי בקומפיילר).

ההבדל בין קריאה לפונקצית מחלקה לא וירטואלית לקריאה לפונקצית מחלקה וירטואלית (בעזרת מצביע):

קריאה לפונקציה לא וירטואלית	קריאה לפונקציה וירטואלית
הפונקציה שצריכה להתבצע ידועה כבר בזמן קומפילציה, ולכן גם כתובתה ידועה	הפונקציה שצריכה להתבצע לא ידועה בזמן קומפילציה, כי היא תלויה בסוג האובייקט שבאמצעותו היא נקראת
	עבור כל מחלקה, הקומפיילר מייצר טבלה שמכילה את הכתובות של הפונקציות הווירטואליות של אותה מחלקה
	כל אובייקט מקבל את הכתובת של הטבלה של המחלקה שאליה הוא שייך, כאשר הוא נוצר (כלומר, בזמן ריצה)
עבור כל קריאה לפונקציה, הקומפיילר מייצר קוד של קפיצה לכתובת של אותה פונקציה	עבור כל קריאה לפונקציה, הקומפיילר מייצר קוד של קפיצה לאחת מהכתובות שרשומות בטבלה של האובייקט

קריאה לפונקציה הלא וירטואלית void MyClass::Func1( ) בעזרת המצביע MyClass* myPtr :

```
Ax = myPtr                    //Get the address of the instance
Bx = MyClass::Func1           //Get the address of the function
Push Ax                       //Push the address of the instance into the stack
Push Bx                       //Push the address of the function into the stack
CallF                         //Save some registers and jump to the beginning of the function
```

קריאה לפונקציה הווירטואלית void MyClass::Func2( ) בעזרת המצביע MyClass* myPtr :

```
Ax = myPtr                    //Get the address of the instance
Bx = myPtr->__vfptr           //Get the address of the instance's V-Table
Cx = Bx + MyClass::Func2_Num  //Add the number of the function in its class
Dx = *Cx                      //Get the address of the appropriate function
Push Ax                       //Push the address of the instance into the stack
Push Dx                       //Push the address of the function into the stack
CallF                         //Save some registers and jump to the beginning of the function
```

כלומר, ביחס לקריאה לפונקציה לא וירטואלית, קריאה לפונקציה וירטואלית דורשת עוד שתי פעולות Read (קריאת הערך של המצביע ל-V-Table וקריאת ערך כלשהו מתוך ה-V-Table) ועוד פעולת Add אחת (חיבור בין רגיסטר למספר קבוע).

תכנות מונחה עצמים:

המונח OOD/OOP לא מתייחס לשפת תכנות כלשהי שמאפשרת דברים ששפת תכנות רגילה לא מאפשרת, אלא מגדיר קונספט שמאפשר לתכנן ולממש תוכנה באופן דומה לתכנון ולמימוש של מערכות אחרות (שאינן בהכרח קשורות לתחום).

הקונספט הזה שם דגש על שני אלמנטים עיקריים:
1. סגירות (Encapsulation).
2. גזירה (Derivation).

הרעיון שעומד מאחורי אלמנט הסגירות, הוא בניית חלקים שונים בתוכנה כמודולים עצמאיים ובלתי תלויים.
אפשר להסתכל על מודול בתוכנה כעל "קופסה שחורה", שמסוגלת לבצע מספר משימות מוגדרות:
1. מה היא מסוגלת לבצע – חשוף בפני כל הצרכנים (המשתמשים).
2. איך היא מבצעת את זה – ידוע רק ליצרן (הממ"מ).

הרעיון הזה מאפשר למספר רב של אנשים לתכנן ולממש חלקים שונים של תוכנה מורכבת, מבלי להצטרך ולהכיר את המבנה הפנימי של כל חלק וחלק. במקום זה, כל אחד צריך להכיר לעומק רק את החלק שלו (מה הוא מסוגל לבצע ואיך הוא מבצע את זה), ואילו את שאר החלקים מספיק לו להכיר מבחוץ (מה הם מסוגלים לבצע). העקרון הוא די דומה לעקרון השימוש בספריות LIB או DLL, למעט העובדה שהן נתונות כקוד מקומפל ולכן המבנה הפנימי שלהן כבר מוסתר מהמשתמש.

הרעיון שעומד מאחורי אלמנט הגזירה, הוא שימוש במודולים קיימים ומוכרים לצורך הרחבתם.
ישנן שתי תבניות כלליות שבעזרתן ניתן להרחיב מודול A למודול B:
1. ליצור את מודול B ולשים בתוכו עצם מסוג A, בנוסף לתכונות הספציפיות שרצויות בו (במקרה הזה, B has an A).
2. לגזור את מודול B ממודול A, ולהוסיף לו את התכונות הספציפיות שרצויות בו (במקרה הזה, B is an A).

הרעיון הזה מאפשר לקחת ממשק מוכר ולהוסיף לו פונקציות חדשות, בלי לממש מחדש את אלה שכבר קיימות. לדוגמא, נניח שמודול A מייצג טייפ שמאפשר Play ו-Stop, ומודול B מייצג טייפ חדשני שמאפשר גם Rewind. פס הייצור של מודול A כבר קיים, ובנוסף, קהל הצרכנים כבר מכיר את הממשק שלו. לכן, במקום ליצור את מודול B מהתחלה, נרחיב את פס הייצור של מודול A ונוסיף לו את מימוש הממשק של מודול B.

כלי נוסף שטמון בשיטת הגזירה, הוא הכללה של קבוצות של מודולים שנגזרים מאותו מקור, ולכן בעלי אופי דומה.
לדוגמא, נניח שמודול A מייצג מכשיר גנרי להשמעת מוזיקה, ומודולים B ו-C מייצגים פטיפון וטייפ בהתאמה:
- ברשותנו אוסף של פטפונים וטייפים, ואנחנו צריכים לבצע לכל מכשיר ניקוי-ראש פעם בשנה.
- פעולת ניקוי הראש יכולה להתבצע רק במעבדה מתאימה, והמעבדה הזולה ביותר היא זאת שמתמחה רק בזה.
- בנוסף, אסור שאף אחד ינסה לבצע שום דבר אחר עם המכשירים, מכיוון שזה עלול להוריד מערכם.

הפתרון המתאים ביותר לבעיה, הוא להגדיר את פעולת ניקוי הראש בממשק של מודול A.
מימוש הפעולה עצמו יכול להתבצע במודול A, או במודולים B ו-C בנפרד (או גם וגם).
הכלי הזה נקרא Polymorphism, והוא מיושם באמצעות הורשה ופונקציות וירטואליות.

לסיכום, שיטת הגזירה מאפשרת להרחיב מודולים קיימים, בלי הצורך לשנות אותם או להכיר את המימוש שלהם.
חשוב לציין שהגמישות הזאת היא לא מושלמת:
על מנת להוסיף מודול חדש לתוכנה קיימת, צריך לקמפל אותה מחדש (כלומר, דרוש קוד המקור של התוכנה).
הדרך היחידה להרחיב תוכנה בלי הצורך לקמפל את קוד המקור שלה מחדש, היא באמצעות שימוש בספריות DLL.

ניהול שכבות הזיכרון:

שכבת הזיכרון הבסיסית ביותר – הרגיסטרים:

- המעבד יכול לגשת אליה ישירות לצורך פעולות של קריאה או כתיבה.
- בנוסף, הוא יכול לבצע עליה את כל הפעולות החישוביות שמוגדרות בו.
- לא ניתן לאחסן בשכבה הזאת מידע, שישמר גם כאשר תיפסק אספקת מתח סדירה.
- מהירות הגישה הממוצעת אל השכבה הזאת היא הגבוהה ביותר (ביחס לשאר שכבות הזיכרון).

מעליה נמצאת שכבת הזיכרון הפנימי (Ram):

- המעבד יכול לגשת אליה ישירות לצורך פעולות של קריאה או כתיבה.
- בנוסף, הוא יכול לבצע עליה חלק מהפעולות החישוביות שמוגדרות בו.
- לא ניתן לאחסן בשכבה הזאת מידע, שישמר גם כאשר תיפסק אספקת מתח סדירה.
- בדרך כלל, השכבה הזאת מורכבת משני חלקים:
  1. ה-Cache (נקרא לעיתים גם L1), שממוקם קרוב למעבד ולכן מהירות הגישה אליו גבוהה יותר.
  2. שאר ה-Ram (נקרא לעיתים גם L2), שממוקם רחוק מהמעבד ולכן מהירות הגישה אליו נמוכה יותר.

מעליה נמצאת שכבת הזיכרון המישני (Hard Disk):

- המעבד לא יכול לגשת אליה ישירות לצורך פעולות של קריאה או כתיבה.
- בנוסף, הוא לא יכול לבצע עליה אף אחת מהפעולות החישוביות שמוגדרות בו.
- ניתן לאחסן בשכבה הזאת מידע, שישמר גם כאשר תיפסק אספקת מתח סדירה.
- מהירות הגישה הממוצעת אל השכבה הזאת היא הנמוכה ביותר (ביחס לשאר שכבות הזיכרון).

על מנת שרצף פעולות (קטע קוד) כלשהו יוכל לרוץ על המעבד, הוא חייב להיטען לזיכרון הפנימי.
המונח זיכרון וירטואלי מתאר צורת ניהול מתוחכמת של הזיכרון הפנימי, תוך שימוש בזיכרון המישני.
מיפוי כתובות הזיכרון הווירטואלי לכתובות בזיכרון הפיזי מתבצע בעזרת יחידת ניהול הזיכרון (MMU).

המטרה העיקרית של הזיכרון הווירטואלי היא לאפשר בכל רגע נתון הרצה של תוכנית, גם כאשר היא דורשת יותר מקום מסך כל הזיכרון הפנימי הפנוי באותו רגע. השיטה היא פשוטה – על מנת שקטע קוד כלשהו יוכל לרוץ על המעבד, הדבר היחידי שחייב להיות בזיכרון הפנימי בכל רגע הוא הפעולה הבאה לביצוע:

- כאשר יש צורך בנתון כלשהו, יחידת ניהול הזיכרון מחפשת אותו ב-Cache.
- אם הנתון לא נמצא שם (Miss), יחידת ניהול הזיכרון מחפשת אותו ב-Ram.
- אם הנתון לא נמצא שם (Page Fault), יחידת ניהול הזיכרון מחפשת אותו ב-Hard Disk.

בכל שלב, במידה והנתון לא נמצא במקום המתאים, יחידת ניהול הזיכרון "משדרגת" את המיקום שלו לפי יוריסטיקה כלשהי (למשל, Last Recently Used), ע"מ לשפר את זמן הגישה אליו בעתיד. דבר זה בא, כמובן, על חשבון נתון אחר. כאשר אין מספיק מקום בזיכרון הפנימי, יחידת ניהול הזיכרון מוציאה נתונים באופן זמני לזיכרון המישני, עד אשר יהיה צורך בהם.

מטרה נוספת של הזיכרון הווירטואלי היא להסיר מהמשתמש את הצורך להכיר כתובות זיכרון אבסולוטיות ולעבוד איתן.
המשתמש הפוטנציאלי יכול להיות המתכנת, הקומפיילר ואפילו מערכת ההפעלה.
הגישה של כל אלה לזיכרון יכולה להתבצע בצורה "שקופה", ורק בזמן ביצוע פעולה שדורשת גישה לזיכרון, ה-MMU נעזר ברגיסטר ה-Base Address ובמפת הזיכרון הווירטואלי ע"מ לחשב את כתובת היעד המדויקת.
בנוסף, העובדה שעבור כל תהליך מוגדר מרחב זיכרון משלו, מאפשרת למערכת ההפעלה להגן מפני גישה של תהליכים אחרים לאותו מרחב זיכרון (שוב, תוך שימוש ברגיסטר ה-Base Address ובמפת הזיכרון הווירטואלי).

ניהול תהליכים וחוטים:

תהליך (Process) הוא למעשה תוכנית בביצוע. כאמור, הגנה על תהליך מפני תהליכים אחרים מתבצעת בעזרת יחידת ניהול הזיכרון, שממפה את כתובות הזיכרון הווירטואלי לכתובות בזיכרון הפיזי. עבור כל תהליך ישנם רגיסטרים, שבעזרתם מוגדר מרחב הזיכרון הפיזי שלו. בכל פעולת גישה של תהליך כלשהו לזיכרון, כתובת היעד מחושבת באמצעות ערכי הרגיסטרים שלו וטבלת הזיכרון הווירטואלי, כך שלמעשה הוא לא יכול לגשת למרחב הזיכרון הפיזי של אף תהליך אחר.

על מנת לדמות סביבה שבה דברים מתבצעים במקביל (Multi Tasking), מערכת ההפעלה מחליפה כל הזמן בין תהליכים. בכל רגע נתון, רק תהליך אחד נמצא במצב ריצה, וכל שאר התהליכים שקיימים במערכת נמצאים במצב המתנה. החלפה בין תהליכים דורשת שימור של המצב הנוכחי עבור התהליך שיוצא (עובר מריצה להמתנה), ושחזור של המצב הקודם עבור התהליך שנכנס (עובר מהמתנה לריצה). החלפה כזאת נקראת החלפת הקשר (Context Switch), והיא כוללת את התוכן של מרחב הזיכרון ואת ערכי הרגיסטרים.

כל תהליך מורכב מחוט (Thread) אחד או יותר, אותם הוא יוצר במהלך הריצה שלו. מערכת ההפעלה מבצעת החלפת הקשר בין חוטים, בדומה להחלפת הקשר בין תהליכים. בצורה כזאת, התהליך (שמייצג תוכנית כלשהי) יכול בעצמו לדמות ביצוע של מספר דברים במקביל. החלפת הקשר בין חוטים של אותו תהליך היא יותר מצומצמת מהחלפת הקשר בין תהליכים שונים. היא לא כוללת את התוכן של מרחב הזיכרון, מכיוון שמרחב הזיכרון הזה משותף לכל החוטים של התהליך.

כאמור, אין צורך לבנות תוכנית, כך שהתהליך שמייצג אותה יהיה מוגן מפני תהליכים אחרים – מערכת ההפעלה דואגת לזה. לעומת זאת, בבניית תוכנית שיוצרת חוטים, יש צורך לסנכרן ביניהם, כך שהגישה שלהם לזיכרון (שמשותף לכולם) תהיה סדירה ודטרמיניסטית. בגלל שמערכת ההפעלה שולטת על זמן הכניסה של כל חוט לריצה ועל זמן היציאה שלו ממנה, לא ניתן לדעת מתי כל חוט (ביחס לחוט אחר של אותו תהליך) ניגש לזיכרון המשותף, ולכן צריך לתזמן את הגישה הזאת.

לעיתים קיימים קטעי קוד (רצף של פעולות) אשר חוטים מסוימים מריצים, ובמהלכם הם ניגשים לאותו אזור בזיכרון המשותף. אם אחד מהחוטים מבצע גישה לצורך כתיבה, אז קטע הקוד של כל אחד מהם מוגדר קריטי (Critical Section). המשמעות היא, שכאשר חוט כלשהו נמצא במהלך הקטע הקריטי שלו, אסור לחוטים האחרים לבצע את הקטע הקריטי שלהם. לצורך כך, יש לממש מנגנון שיבטיח מניעה הדדית (Mutual Exclusion). המנגנון שממומש, חייב גם למנוע מצב שבו קבוצה של חוטים נמצאת בהמתנה "ציקלית" לכניסה של אחד מהם לקטע הקריטי (מניעת Deadlock). בנוסף, הוא גם חייב למנוע מצב שבו חוט כלשהו ממתין לנצח להתחיל את הקטע הקריטי שלו (מניעת Starvation).

שיטות בסיסיות להגנה על קטע קוד קריטי:

1. <u>Disable/Enable Interrupts</u> – נגדיר את הקטע הקריטי של כל חוט כפעולה אטומית (רצף של פעולות, אשר בתחילתו הפסיקות מנוטרלות ובסופו הן מאופשרות, ולכן במהלכו לא יכולה להתבצע החלפת הקשר). הבעיה היא, שבזמן שחוט כלשהו יבצע את הקטע הקריטי שלו, אף חוט אחר לא יוכל לרוץ (גם לא לצורך ביצוע של פעולות מחוץ לקטע הקריטי).
2. <u>Busy Wait</u> – נגדיר משתנה x=1 בזיכרון המשותף (משתנה גלובאלי). כל חוט שיגיע לקטע הקריטי שלו:
   - ינטרל את הפסיקות.
   - אם x שווה ל-0, יאפשר את הפסיקות ויחזור לשלב הקודם.
   - ישנה את x ל-0, יאפשר את הפסיקות, יבצע את הקטע הקריטי ויחזיר את x ל-1.

   הבעיה היא, שחוט יכול לעבור מהמתנה לריצה רק בשביל "לגלות" שהערך של x הוא עדיין 0.
   מצב כזה יכול לחזור על עצמו הרבה פעמים, כשבכל פעם מתבצעת החלפת הקשר ומתבזבז זמן.
3. <u>Mutex</u> – נגדיר אובייקט Mtx, עם ממשק של שתי פעולות אטומיות:
   - Mtx.Wait( ) – אם Mtx משוחרר, נעל אותו. אחרת, הוצא את החוט הנוכחי להמתנה.
   - Mtx.Signal( ) – אם Mtx נעול, שחרר אותו והוצא מהמתנה את החוט שנמצא בה הכי הרבה זמן.

   כל חוט יבצע את הפעולה Mtx.Wait( ) בתחילת הקטע הקריטי שלו, ואת הפעולה Mtx.Signal( ) בסופו.
   האובייקט Mtx ימוקם בזיכרון המשותף, וכך יוכר ע"י כל החוטים. בנוסף, הוא ישמור רשימה של החוטים שהוא הוציא להמתנה. בניגוד לשיטת ה-Busy Wait, החסימה של החוט בתחילת הקטע הקריטי לא מתבצעת בקוד של החוט עצמו ע"י המתנה "על" משתנה גלובאלי. במקום זה, היא מתבצעת ע"י בקשה ממערכת ההפעלה להוציא את החוט להמתנה. היתרון בכך הוא, שאין צורך לבצע החלפת הקשר על מנת ש"קוד ההמתנה" של החוט ירוץ במעבד.
   <u>הסבר:</u> מה שבעצם קורה כאשר Mtx מוציא חוט להמתנה, הוא שהחוט קורא לפונקציה, שבסופה הוא מבקש ממערכת ההפעלה להוציא אותו להמתנה (קריאת מערכת Block). כך שלמעשה, בעזרת מנגנון ה-Mutex, כל חוט שמגיע לקטע הקריטי כאשר חוט אחר כבר נמצא בו, מוציא <u>את עצמו</u> להמתנה. חוט שמסיים את הקטע הקריטי, מבקש ממערכת ההפעלה להוציא מהמתנה את החוט שנמצא בה הכי הרבה זמן (קריאת מערכת Wakeup).

תשובה לשאלה 1

שלב ראשון – נעביר את כל הכדורים האדומים לתחילת המערך:

1. נחזיק אינדקס A שמצביע על תחילת המערך ואינדקס B שמצביע על סוף המערך.
2. נקדם את אינדקס A לכיוון סוף המערך, עד שניתקל בכדור שאינו אדום.
3. נקדם את אינדקס B לכיוון תחילת המערך, עד שניתקל בכדור אדום.
4. אם אינדקס A עבר את אינדקס B, נעצור את האלגוריתם.
5. נחליף בין הכדורים שמוצבעים ע"י האינדקסים.
6. נחזור על התהליך (החל מסעיף 2).

שלב שני – באופן דומה, נעביר את כל הכדורים הירוקים לסוף המערך...

סיבוכיות זמן – O(N), סיבוכיות זיכרון – O(1).

תשובה לשאלה 2

על מנת להפוך רשימה מקושרת חד-כיוונית:

```
void Node::Swap(Node* pNext)
{
	if (pNext != NULL)
	{
		pNext->Swap(pNext->m_pNext);
		pNext->m_pNext = this;
	}
}

void List::Reverse()
{
	if (m_pHead != NULL)
	{
		m_pHead->Swap(m_pHead->GetNext());
		m_pHead->SetNext(NULL);
	}
	Node* pTemp = m_pHead;
	m_pHead = m_pTail;
	m_pTail = pTemp;
}
```

על מנת להפוך רשימה מקושרת דו-כיוונית:

```
void Node::Swap()
{
	if (m_pNext != NULL)
	{
		m_pNext->Swap();
		m_pNext->m_pNext = this;
	}
	m_pPrev = m_pNext;
}

void List::Reverse()
{
	if (m_pHead != NULL)
	{
		m_pHead->Swap();
		m_pHead->SetNext(NULL);
	}
	Node* pTemp = m_pHead;
	m_pHead = m_pTail;
	m_pTail = pTemp;
}
```

תשובה לשאלה 3

נשתמש בשני גלאים שמונחים בצורה טורית: גלאי A לפני הכניסה לאולם וגלאי B אחרי הכניסה לאולם.
עבור כל גלאי נחזיק משתנה בוליאני f, שמאותחל ל-0.
בנוסף, נחזיק מונה של מספר האנשים שבתוך האולם בכל רגע נתון.
כאשר A מזהה תנועה:

- אם f(B)=0, אז התנועה היא כלפי פנים. נאתחל את f(A) ל-1.
- אם f(B)=1, אז התנועה היא כלפי חוץ. נחסיר 1 מהמונה, ונאפס את f(A) ואת f(B).

כאשר B מזהה תנועה:

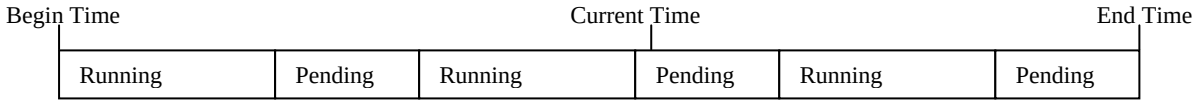
- אם f(A)=0, אז התנועה היא כלפי חוץ. נאתחל את f(B) ל-1.
- אם f(A)=1, אז התנועה היא כלפי פנים. נוסיף 1 למונה, ונאפס את f(A) ואת f(B).

בעיה אפשרית עלולה להיווצר כאשר מישהו משנה את דעתו באמצע הדרך (בין הגלאים) וחוזר.

תשובה לשאלה 4

כאשר הזמן הנוכחי לא נמצא בין זמן ההתחלה לזמן הסיום של ה-Task, ה-Task לא קיים.
כאשר הזמן הנוכחי כן נמצא בין זמן ההתחלה לזמן הסיום של ה-Task, נבצע את הבדיקה הבאה:

- נחשב את השארית בין ההפרש [זמן נוכחי פחות זמן התחלה] לסכום [אורך זמן ריצה ועוד אורך זמן המתנה].
- אם השארית שקיבלנו היא קטנה מאורך זמן הריצה של ה-Task, אז הוא במצב ריצה. אחרת, הוא במצב המתנה.




```
STATE Task::GetState(iCurrentTime)
{
	if (m_iBeginTime<=iCurrentTime && iCurrentTime<=m_iEndTime)
	{
		int iRem=(iCurrentTime-m_iBeginTime)%(m_iRunTime+m_iPendTime);
		if (iRem<m_iRunTime)
			return RUNNING;
		else
			return PENDING;
	}
	else
		return NOT_ALIVE;
}
```

ההנחה היא כמובן, שה-Task תמיד מתחיל במצב ריצה.
אם הוא מתחיל במצב המתנה, אז צריך להחליף את התנאי iRem<m_iRunTime בתנאי iRem>=m_iPendTime.

תשובה לשאלה 5

כדי לדעת כמה אובייקטים מסוג מסוים קיימים בכל רגע נתון, צריך להוסיף למחלקה שלהם משתנה סטאטי שסופר אותם.
בכל Constructor של המחלקה צריך להגדיל את המשתנה הזה ב-1, וב-Destructor שלה צריך להקטין אותו ב-1.

תשובה לשאלה 6

המילה השמורה static מייצגת "תכונה", אשר ניתן לאפיין בעזרתה משתנים ופונקציות:

- משתנה סטאטי יכול להופיע ברמת הפונקציה (משתנה לוקאלי סטאטי), ברמת הקובץ (משתנה גלובאלי סטאטי) וברמת המחלקה (משתנה מחלקתי סטאטי). סוג נוסף ששייך לאותה קטגוריה, הוא משתנה שמופיע ברמת התוכנית (משתנה גלובאלי לא סטאטי). ההבדל בין ארבעת הסוגים השונים הוא רק ב-Scope שבו הקומפיילר מכיר אותם. בזמן ריצה, משתנים כאלה נמצאים ב-Data Section ומיקומם נשאר קבוע (זאת, בניגוד למשתנים לוקאליים לא סטאטיים, שבזמן ריצה נמצאים ב-Stack ומיקומם עשוי להשתנות).
- פונקציה סטאטית יכולה להופיע ברמת הקובץ (פונקציה גלובאלית סטאטית) וברמת המחלקה (פונקציה מחלקתית סטאטית). סוג נוסף ששייך לאותה קטגוריה, הוא פונקציה שמופיעה ברמת התוכנית (פונקציה גלובאלית לא סטאטית). ההבדל בין שלושת הסוגים השונים הוא רק ב-Scope שבו הקומפיילר מכיר אותם. פונקציות כאלה מקבלות כפרמטרים רק את מה שמוגדר עבורן (זאת, בניגוד לפונקציות מחלקתיות לא סטאטיות, שכל אחת מהן מקבלת כפרמטר גם את כתובת האובייקט שבאמצעותו היא נקראת).

לפירוט נוסף – חלק 3 (חומר כללי: סמלים בתוכנה).

תשובה לשאלה 7

נבצע N איטרציות.

באיטרציה מספר i:

1. ברשימת הקלט, נבחר שיר אקראי מתוך N-i השירים הראשונים.
2. ברשימת הפלט, נכניס אותו למקום ה-i.
3. ברשימת הקלט, נחליף בינו לבין השיר שנמצא במקום ה-(N-i), ע"מ שלא נוכל לבחור אותו באיטרציה הבאה.

```
void RandReorder(Song* inputArray[N], Song* outputArray[N])
{
	for (int i=0; i<N; i++)
	{
		int index = Rand()%(N-i);
		outputArray[i] = inputArray[index];
		Swap(inputArray,index,N-1-i);
	}
}
```

שיטה נוספת:

באופן חד-פעמי, נחשב רשימה של כל המספרים בתחום (1,N) אשר אין להם אף מחלק משותף עם N.

בכל פעם שנרצה לסדר את השירים מחדש:

1. נשתמש בפונקציה Rand ע"מ לבחור מספר אקראי מהרשימה החלקית שחישבנו. נקרא לו P.
2. נתחיל משיר מספר P (ברשימת השירים המלאה), ונתקדם P שירים בכל איטרציה, במשך N איטרציות.
3. את ההתקדמות נבצע בצורה ציקלית: index = (index + P) % N .

לפי חוק הציקליות בחבורות (או משהו כזה), אנחנו נעבור כל שיר פעם אחת בדיוק לפני שנחזור לשיר הראשון.

בעיות אפשריות:

א. זמן חישוב הרשימה החלקית עלול להיות ארוך (למרות שזה לא עקרוני, כי מחשבים אותה רק פעם אחת).

ב. על פני הרבה הרצות של הפונקציה, לא נקבל את כל הסידורים האפשריים של N שירים בהתפלגות אחידה (כמו בפתרון הראשון). למעשה, חלק גדול מהסידורים האפשריים לא נקבל כלל. הסיבה לכך היא:
- מספר הסידורים השונים שנוכל לקבל בשיטה הזאת = גודל הרשימה החלקית < N.
- מספר הסידורים השונים האפשרי עבור N שירים = N * … * 2 * 1 = N עצרת.

תשובה לשאלה 8

על מנת להבטיח שאף אחד לא יוכל ליצור באופן ישיר עצם מהסוג של מחלקה כלשהי, ישנן שתי אפשרויות:

א. לשים את ה-Constructor ב-protected של המחלקה. ניתן יהיה ליצור עצמים של מחלקות שיורשות ממנה.

ב. לשים את ה-Constructor ב-private של המחלקה, ולהוסיף לה פונקציה סטאטית, שיוצרת עצם מהסוג שלה ומחזירה מצביע אליו (בדומה לשיטת ה-Singleton, למעט העובדה שזה לא חייב להיות העצם היחידי מסוגה).

תשובה לשאלה 9

המנורות שיישארו דלוקות: אלה שמספר המחלקים של האינדקס שלהן הוא אי-זוגי.

האינדקסים שמספר המחלקים שלהם הוא אי-זוגי: מספרים ריבועיים – 1, 4, 9, 16...

תשובה לשאלה 10

אם לא ידוע מי מתחיל, אז אין שיטה כזאת, כי זה פרדוקס (אם הייתה שיטה כזאת, אז יש סיכוי שהשחקן השני היה משתמש בה ומנצח – סתירה).

השחקן הראשון שמתחיל יכול לנצח:

א. את המטבע הראשון הוא צריך לשים במרכז השולחן.

ב. עבור כל מטבע שהיריב שלו שם, הוא צריך לשים מטבע בצד הנגדי (בקו ישר ובמרחק שווה מהמרכז).

תשובה לשאלה 11

צריך לשים כדור אדום אחד בסל בראשון, ואת כל שאר הכדורים (אדומים ושחורים) בסל השני.
נחשב את ההסתברויות הבאות:

- A = ההסתברות לגשת לסל הראשון = 1/2
- B = ההסתברות להוציא כדור אדום מהסל הראשון = 1/1
- C = ההסתברות לגשת לסל השני = 1/2
- D = ההסתברות להוציא כדור אדום מהסל השני = 9/19

נקבל את התוצאה:

- A*B+C*D = ההסתברות להוציא כדור אדום = 14/19

תשובה לשאלה 12

נניח שמספר הניסיונות שצריך לבצע בשביל לגלות באיזה קומה הכדורים נשברים, הוא לכל הפחות N.
על מנת להבטיח שנבצע לכל היותר N ניסיונות, נשתמש באלגוריתם הבא (החל מ k=N ועד ל k=1):

1. נעלה k קומות ונזרוק את הכדור הראשון. אם הוא נשבר, נרד k-1 קומות:
   - נזרוק את הכדור השני. אם הוא נשבר: גילינו את מספר הקומה.
   - אחרת, נעלה קומה אחת ונחזור על התהליך.
2. אחרת, נקטין את k ב-1 ונחזור על התהליך.

הסבר:

- בשלב הראשון נזרוק את הכדור הראשון מקומה N.
  - אם הוא נשבר, ננסה את הכדור השני החל מקומה 1 ועד לקומה N-1 (או עד שהוא יישבר).
  - סה"כ, ביצענו לכל היותר N ניסיונות (פעם עבור הכדור הראשון, N-1 פעמים עבור הכדור השני).
- בשלב השני נזרוק את הכדור הראשון מקומה N+N-1.
  - אם הוא נשבר, ננסה את הכדור השני החל מקומה N+1 ועד לקומה N+N-2 (או עד שהוא יישבר).
  - סה"כ, ביצענו לכל היותר N ניסיונות (פעמיים עבור הכדור הראשון, N-2 פעמים עבור הכדור השני).
- בכל שלב נעלה קומה אחת פחות ממה שעלינו בשלב הקודם, ולכן מספר הניסיונות שנבצע לכל היותר יישאר N.

חישוב הערך של N עבור בניין בן 100 קומות:

- ע"מ לוודא שלאחר N ניסיונות נגיע לפחות לקומה 100, צריך שיתקיים: $N + N-1 + N-2 + \ldots + 1 \geq 100$
- $(N+1)*N/2 \geq 100$ ➔ $N^2+N-200 \geq 0$ ➔ $N \geq [-1+\sqrt{(1+800)}]/2$ ➔ $N \geq 13.65$ ➔ $N = 14$
- כלומר, לכל היותר נזרוק את הכדור הראשון מהקומות הבאות: 14, 27, 39, 50, 60, 69, 77, 84, 90, 95, 99.

תשובה לשאלה 13

את בית מספר 1 נצלם תמיד (כדי שבכל מקרה תהיה לנו תמונה, אם אין אחריו עוד בית).
את בית מספר 2 נצלם בהסתברות 1/2, תוך שימוש בפונקציה Rand()%2.
את בית מספר 3 נצלם בהסתברות 1/3, תוך שימוש בפונקציה Rand()%3.
את בית מספר X נצלם בהסתברות 1/X, תוך שימוש בפונקציה Rand()%X.
נוכיח באינדוקציה שעבור N בתים, ההסתברות לקבל תמונה של כל אחד ואחד מהם היא זהה:

- עבור N=1, בית מספר 1 מתקבל בהסתברות של 100%.
- עבור N=2, בית מספר 2 מתקבל בהסתברות של 50%, ולכן ההסתברות של בית מספר 1 יורדת ל-50%.
- נניח נכונות עבור N=k ונוכיח נכונות עבור N=k+1:
  - לפי הנחת האינדוקציה עבור k בתים, ההסתברות לתמונה של כל אחד ואחד מהם היא זהה. ערכה של ההסתברות הזאת הוא 1/k.
  - לפי האלגוריתם, את בית מספר k+1 נצלם בהסתברות 1/(k+1), ולכן ההסתברות שתישאר בידינו תמונה של אחד מ-k הבתים הקודמים היא 1-1/(k+1), כלומר k/(k+1).
  - מכיוון שעבור k הבתים הקודמים, ההסתברות לתמונה של כל אחד ואחד מהם היא עדיין זהה, נקבל שערכה של ההסתברות הזאת הוא עכשיו [k/(k+1)]/k, כלומר 1/(k+1).
  - מסקנה – ההסתברות לתמונה של בית מספר k+1 זהה להסתברות לתמונה של כל אחד מ-k הבתים הקודמים. ערכה של ההסתברות הזאת הוא 1/(k+1).

תשובה לשאלה 14

אותו עקרון כמו בתשובה לשאלה 7.

תשובה לשאלה 15

פונקציות וירטואליות הן הכלי שבעזרתו (ובעזרת הורשה) ניתן ליישם Polymorphism ב-C++:

- הפונקציה שצריכה להתבצע לא ידועה בזמן קומפילציה, כי היא תלויה בסוג האובייקט שבאמצעותו היא נקראת.
- עבור כל מחלקה, הקומפיילר מייצר טבלה שמכילה את הכתובות של הפונקציות הווירטואליות של אותה מחלקה.
- עבור כל קריאה לפונקציה, הקומפיילר מייצר קוד של קפיצה לאחת מהכתובות שרשומות בטבלה של האובייקט.
- כל אובייקט מקבל את הכתובת של הטבלה של המחלקה שאליה הוא שייך, כאשר הוא נוצר (כלומר, בזמן ריצה).

לפירוט נוסף – חלק 3 (חומר כללי: פונקציות מחלקה).

תשובה לשאלה 16

מימוש פונקציות עם מספר לא קבוע של ארגומנטים הוא דבר שימושי ב-C.
ב-C++, אפשר להשתמש בערכי Default עבור הארגומנטים שהפונקציה מקבלת.
חתימה של פונקציה עם מספר לא קבוע של ארגומנטים נראית כך: void func(int x, …) .
הערך המוחזר של הפונקציה יכול להיות מכל סוג שהוא, וחייב להיות לפחות ארגומנט אחד מוצהר.
לאחר הארגומנטים המוצהרים באות שלוש נקודות (...), שמעידות על מספר לא קבוע של ארגומנטים.
בכל פעם שמתבצעת קריאה לפונקציה (במהלך ריצת התוכנית), הארגומנטים מועברים אליה במחסנית.
במימוש של הפונקציה, על מנת לגשת אל הארגומנטים שמועברים אליה אבל לא מוצהרים בחתימה שלה:

- בודקים את הכתובת של הארגומנט המוצהר האחרון.
- הארגומנט הבא ממוקם 4 או 8 בתים אחריו, וכך גם לגבי כל ארגומנט נוסף שנשלח לפונקציה.
- ארגומנט שגודלו 4 בתים או פחות נפרש על פני 4 בתים, וארגומנט שגודלו יותר נפרש על פני 8 בתים.
- חשוב לזכור, שכאשר שולחים לפונקציה מערך, היא מקבלת את הכתובת שלו בזיכרון ולא את הערכים עצמם.
  למשל, אם שולחים לפונקציה מחרוזת, אז היא מקבלת את הכתובת בזיכרון שבה נמצאים התווים של המחרוזת.

לדוגמא, עבור הקריאה הזאת לפונקציה:

```
short y = 1000;
int sum = func(1,y,5000,"abc");
```

אפשר לממש את הפונקציה כך:

```
int func(char x, …)
{
        short   y = (short)((int*)&x+1)[0];   //y = 1000
        int     z = (int   )((int*)&x+2)[0];   //z = 5000
        char*   s = (char*)((int*)&x+3)[0];   //s[0…2] = "abc"
        return x+y+z+s[0];                     //1+1000+5000+'a' = 6098
}
```

אפשר לממש את זה בצורה יותר נוחה בעזרת va_list, va_start, va_end, va_arg (מוגדרים בקובץ stdarg.h).
החסרון העיקרי בפונקציה מהסוג הזה:
מכיוון שסוגי הארגומנטים ומספרם לא ידועים, מי שמממש אותה צריך "לנחש" אותם.
בפונקציה printf למשל, זיהוי סוגי הארגומנטים ומספרם מתבצע לפי המחרוזת שהיא מקבלת (בארגומנט הראשון):

- היא מניחה שמספר הארגומנטים שנשלחו אליה הוא מספר הפעמים שהתו '%' מופיע במחרוזת.
- היא מנחשת מהו הסוג של כל ארגומנט לפי האות שמופיעה ליד כל אחד מתווי ה-'%' במחרוזת.

הבעיה היא, שאם בקריאה לפונקציה היא מקבלת פחות ארגומנטים ממה שהיא מניחה שהיא קיבלה, היא ניגשת למקום בזיכרון שאין לו משמעות ברורה ("זבל"). בנוסף, כאשר גודל הארגומנט שנשלח שונה מהגודל שהיא מניחה שהוא, היא "יוצאת מסינכרון" לגביו ולגבי כל שאר הארגומנטים שבאים אחריו (הגודל יכול להיות 4 בתים או 8 בתים).
דוגמאות:

א. הרצה של printf("%d %d",5) :
   - הפונקציה ניגשת לאזור שלא הוגדר עבורה בזיכרון וקוראת "זבל" (או מפילה את התוכנית).

ב. הרצה של printf("%d %f",1.2,3.4) :
   - הפונקציה מקבלת שני ארגומנטים בגודל 8 בתים ובפורמט Floating-Point כל אחד.
   - מ-4 הבתים הראשונים היא קוראת מספר בפורמט Fixed-Point.
   - מ-8 הבתים הבאים היא קוראת מספר בפורמט Floating-Point.
   - מ-4 בתים האחרונים היא מתעלמת.

מכיוון שסוגי הארגומנטים ומספרם לא ידועים גם לקומפיילר, הוא לא ייתן אזהרות או שגיאות קומפילציה עבור קריאה לפונקציה כזאת עם ארגומנטים שאינם "חוקיים" מבחינתה (כמו בדוגמאות הנ"ל). התוצאה – שגיאות בזמן ריצה.

תשובה לשאלה 17

על מנת לסנכרן בין שני החוטים, נבצע Busy Wait באופן הבא:

1. נגדיר משתנה לוקאלי y=1, שבעזרתו נדע את מספר האיטרציה שבה נמצא כל חוט.
2. נגדיר משתנה גלובאלי x=0, ונבצע עליו Busy Wait ב"נקודת ההמתנה" של כל חוט.

```
for (int y=1; ; y++)
{
        …
        …
        DisableInterrupts();
        x++; //Load x, Inc, Store x
        EnableInterrupts();
        while (x < y*2) { }
        …
        …
}
```

הסבר:

- החוט הראשון שיגיע ל"נקודת ההמתנה" באיטרציה מספר y, יגדיל את x ב-1, ומיד לאחר מכן "ימתין" לחוט השני.
- החוט השני שיגיע ל"נקודת ההמתנה" באיטרציה מספר y, יגדיל את x ב-1, ועל ידי כך "ישחרר" את החוט הראשון.

הערות:

א. השימוש ב-Busy Wait עלול להיות בזבזני, מכיוון שבזמן שהחוט הראשון ממתין שהחוט השני יגיע לאותה נקודה, יכולות להתבצע החלפות הקשר רק לצורך בדיקת הערך של x (בקוד של החוט הראשון).

ב. פעולת ההגדלה של x מורכבת מרצף של מספר פקודות מכונה, וע"מ שבמהלכה לא תוכל להתבצע החלפת הקשר, היא מוגדרת כפעולה אטומית (ע"י ניטרול הפסיקות בתחילתה ואיפשור הפסיקות בסופה).

עבור N חוטים אפשר לבצע Busy Wait באופן דומה, אבל השימוש בו יהיה הרבה יותר בזבזני מבחינת זמן, מכיוון שהפעם ייתכנו החלפות הקשר לצורך בדיקת הערך של x, בקוד של כל אחד מהחוטים שממתינים שהערך הזה יגיע ל-N.
לכן, על מנת לסנכרן בין N חוטים, נבצע Priority Inversion באופן הבא:

```
while (true)
{
        …
        …
        DisableInterrupts();
        if (x < N-1)
        {
                DecreaseThisThreadPriority();
                x++;
        }
        else
        {
                IncreaseAllThreadsPriority();
                x = 0;
        }
        EnableInterrupts();
        …
        …
}
```

הסבר:

- למעט החוט האחרון, כל חוט שיגיע ל"נקודת ההמתנה", יוריד לעצמו את העדיפות. לאחר מכן, הוא יאפשר את הפסיקות ויעבור מיד להמתנה (מכיוון שיהיה חוט אחר בעדיפות גבוהה יותר).
- החוט האחרון שיגיע ל"נקודת ההמתנה", יעלה לכל החוטים האחרים את העדיפות. לאחר מכן, הוא יאפשר את הפסיקות והם יוכלו להמשיך לרוץ (מכיוון שכל החוטים יהיו בעדיפות שווה).

תשובה לשאלה 18

נסמן מחרוזת באורך של n תווים באופן הבא: S[1…n].

נממש פונקציה רקורסיבית, שמקבלת מחרוזת S[1…n] ומחזירה את רשימת כל הפרמוטציות שלה:

- תנאי העצירה: כאשר מחרוזת הקלט ריקה, נחזיר רשימה שבה מחרוזת אחת (ריקה).
- צעד רקורסיה מספר x:
  1. נקרא לפונקציה עם תת המחרוזת S[x+1…n], ע"מ לקבל את רשימת כל הפרמוטציות שלה.
  2. נקבל רשימה באורך של !(n-x) מחרוזות, כל אחת באורך של n-x תווים.
  3. נעבור על הרשימה, ולכל מחרוזת נוסיף את התו S(x) בכל מקום אפשרי.
  4. לכל מחרוזת יש לנו n-x+1 מקומות אפשריים להוסיף את התו S(x).
  5. נקבל רשימה חדשה באורך של !(n-x+1) מחרוזות, כל אחת באורך של n-x+1 תווים.

```
List Permute(String string)
{
	if (string.length == 0)
		return EmptyStrings(1);
	List prevList = Permute(SubString(string,1,string.length));
	List nextList = EmptyStrings(string.length*prevList.length);
	for (int i=0; i<prevList.length; i++)
	{
		for (int j=0; j<string.length; j++)
		{
			nextList[i*string.length+j] += SubString(prevList[i],0,j);
			nextList[i*string.length+j] += string[0];
			nextList[i*string.length+j] += SubString(prevList[i],j,string.length-1);
		}
	}
	return nextList;
}
```

הערות:

א. הפונקציה EmptyStrings(n) תחזיר רשימה של n מחרוזות ריקות.

ב. הפונקציה SubString(str,i,j) תחזיר תת מחרוזת של str, החל מאינדקס i (כולל) ועד לאינדקס j (לא כולל).

תשובה לשאלה 19

מציאת האינדקס של המופע הראשון של תו כלשהו מ-Set ב-String:

1. ניעזר במערך בוליאני שגודלו 256 תאים (כמספר התווים בקוד ASCII).
2. נעבור על אוסף התווים Set, ועבור כל תו נרשום true במקום המתאים לו במערך.
3. נעבור על המחרוזת String, ונחפש את התו הראשון אשר במקום המתאים לו במערך רשום true.

```
int FirstIndex(const char* string, const char* set)
{
	bool array[256] = {false};
	for (int i=0; i<strlen(set); i++)
		array[set[i]] = true;
	for (int j=0; j<strlen(string); j++)
		if (array[string[j]] == true)
			return j;
	return -1;
}
```

סיבוכיות הזמן שווה לסכום אורכי הקלט. סיבוכיות הזיכרון היא כמספר התווים האפשריים.
אם התווים יהיו בקוד רחב יותר, נצטרך מערך גדול יותר, ולכן סיבוכיות הזיכרון תהיה גבוהה יותר. במקרה כזה אפשר להשתמש ב-Hash Table יותר מתוחכמת (המערך הוא למעשה מימוש של Hashing ישיר). סביר להניח שכדי למצוא את המקום המתאים לתו כלשהו ב-Hash Table כזאת, נצטרך יותר מפעולה אחת. על מנת להשיג את אותה סיבוכיות זמן, נצטרך לממש טבלה שמאפשרת Hashing בעזרת מספר קבוע של פעולות (וללא כל תלות באורך הקלט). בנוסף, נצטרך לדאוג שגודל הטבלה הזאת לא יהיה תלוי במספר התווים האפשריים.
אפשרות נוספת: פשוט לעבור על התווים ב-String לפי הסדר, ועבור כל תו לבדוק אם הוא נמצא ב-Set. במקרה הזה לא נצטרך זיכרון נוסף בכלל, אבל סיבוכיות הזמן תגדל מסכום אורכי הקלט למכפלת אורכי הקלט.

תשובה לשאלה 20

ע"מ לייצג את המילון בצורה יעילה אפשר להשתמש ב-Hash Table, שכל תא בה מייצג את האות הראשונה במילה.
בנוסף, בכל תא תהיה Hash Table, שכל תא בה מייצג את האות השניה במילה, וכן הלאה באופן דומה.
תא שמייצג את האות האחרונה במילה כלשהי יכיל גם את פירוש המילה.
למעשה, המילון מיוצג באמצעות עצים: לכל אות קיים עץ שמחזיק את כל המילים שמתחילות בה.
סיבוכיות הזמן של הפעולות השונות (הוספת מילה, חיפוש מילה, מחיקת מילה) תלויה בסוג ה-Hash Table שנממש.
סיבוכיות הזיכרון של המילון תלויה גם היא בסוג ה-Hash Table.
לדוגמא, אם נשתמש ב-Hash Table ישירה (מערך), סיבוכיות הזמן תהיה לינארית באורך הקלט.
לעומת זאת, סיבוכיות הזיכרון תהיה מאד בזבזנית, מכיוון שמבנה המילון יאפשר קיום של כל קומבינציה אפשרית בכל אורך שהוא (עד גבול מסוים שיוגדר מראש).
ע"מ למצוא במילון את כל המילים שהן פרמוטציות של מילה מסוימת, ישנן שתי אפשרויות:

א. נשתמש בפונקציה שתוארה בתשובה לשאלה 18, ונבדוק עבור כל פרמוטציה שהפונקציה מחזירה אם היא נמצאת במילון. הבעיה היא, שנצטרך לעבור על כל הפרמוטציות האפשריות של המילה, גם אם רובן לא נמצאות במילון.
ב. נתאים לכל אות מספר ראשוני ייחודי, ולכל מילה את מכפלת המספרים שמייצגים את האותיות שלה. בגלל שכולם ראשוניים, מובטח לנו מספר ייחודי עבור כל מילה והפרמוטציות שלה. נבצע מעבר חד-פעמי על המילון ונשמור כל קבוצת פרמוטציות במערך, כאשר המספר שמייצג את הקבוצה הוא האינדקס שלה במערך. הבעיה היא, שזה לא כל כך מעשי, כי המספרים שמייצגים את המילים יכולים להיות עצומים.

תשובה לשאלה 21

ישנם סה"כ 4096 רצפים אפשריים: 2^12 = 4096. כאמור, כל אופרנד הוא באורך 3 ביטים.
עבור 4 פעולות שמקבלות שלושה אופרנדים כל אחת, דרושים לנו 2048 רצפים שונים: 4 * 2^(3*3) = 2048.
עבור 255 פעולות שמקבלות אופרנד אחד כל אחת, דרושים לנו 2040 רצפים שונים: 255 * 2^(1*3) = 2040.
עבור 16 פעולות שלא מקבלות אופרנדים כלל, דרושים לנו 16 רצפים שונים: 16 * 2^(0*3) = 16.
סה"כ דרושים לנו 4104 רצפים שונים – יותר ממספר הרצפים האפשריים, ולכן לא ניתן לתכנן מעבד כזה.
תנאי עבור המקרה הכללי: אסור שהסכום $\sum[K_i * 2^{(P_i*L_i)}]$ יהיה יותר גדול ממספר הרצפים האפשריים.

תשובה לשאלה 22

קידוד העץ לקובץ:

1. נעבור על העץ בשיטת Pre-Order (אב, בן שמאלי, בן ימני), ולכל צומת ניתן מספר סידורי.
2. נרשום בקובץ את מספר הצמתים שיש בעץ.
3. נקצה מערך של מצביעים, שבו נוכל לרשום את הצמתים בסדר עולה (לפי מספר סידורי).
4. נעבור על העץ שוב, ולכל צומת נרשום במקום המתאים במערך את הכתובת שלו.
5. נעבור על המערך לפי הסדר, ולכל צומת נרשום בקובץ את תוכן הרשומה שלו.
6. נעבור על המערך שוב, ולכל צומת נרשום בקובץ את המספרים הסידוריים של שני הבנים שלו.

הערה – כאשר לצומת יש פחות משני בנים, נרשום 0 (במקום מספר סידורי) עבור כל בן שחסר לו.

פענוח הקובץ לעץ:

1. נקרא מהקובץ את מספר הצמתים שיש בעץ, ונקצה מערך של מצביעים.
2. נקרא מהקובץ רשומות לפי הסדר, לכל אחת ניצור צומת שמכיל אותה, ונרשום את כתובתו של הצומת במערך.
3. נעבור על המערך לפי הסדר, לכל צומת נקרא מהקובץ זוג מספרים, ונקשר את הצומת לאיברים המתאימים במערך.

הערה – מכיוון שקודדנו את העץ בשיטת Pre-Order, כתובתו של השורש תהיה במקום הראשון במערך.

תשובה לשאלה 23

מחלקה אבסטרקטית זאת מחלקה שיש בה פונקציות וירטואליות טהורות, ולכן אי אפשר ליצור עצמים מהסוג שלה.
הפונקציות הווירטואליות של מחלקה כזאת מגדירות ממשק גנרי, אשר מחלקות שנגזרות (יורשות) ממנה צריכות לממש.

תשובה לשאלה 24

פונקציה וירטואלית:

א. במחלקה שבה היא מוגדרת – חייבים לממש אם רוצים לאפשר יצירה של עצמים מסוגה.
ב. במחלקות שיורשות ממנה – לא חייבים לממש (אפשר ליצור עצמים מסוגן בכל מקרה).

פונקציה וירטואלית טהורה:

א. במחלקה שבה היא מוגדרת – לא חייבים לממש (אי אפשר ליצור עצמים מסוגה בכל מקרה).
ב. במחלקות שיורשות ממנה – חייבים לממש אם רוצים לאפשר יצירה של עצמים מסוגן.

תשובה לשאלה 25

Copy Constructor זה פונקצית מחלקה, שמקבלת עצם מסוג המחלקה By Reference. קריאה ל-Constructor הזה מתבצעת כאשר מאתחלים עצם בשורת ההצהרה (לדוגמא, A a2=a1), וכאשר שולחים לפונקציה כלשהי או מחזירים מפונקציה כלשהי עצם By Value (בשני המקרים האלה, ה-Copy Constructor יוצר עצם חדש במחסנית הפונקציה).
כאשר ה-Copy Constructor של מחלקה כלשהי לא מוגדר באופן מפורש ע"י המתכנת, הקומפיילר מגדיר אותו בתור ברירת מחדל. במקרה כזה, ה-Copy Constructor פשוט מעתיק את השדות של העצם שהוא מקבל (לתוך העצם שלו).
אם חלק מהשדות האלה הם מצביעים לזיכרון, אז שני העצמים מתייחסים לאותו אזור בזיכרון ועלולים לבצע עליו את אותן פעולות. לדוגמא, כאשר מדובר בזיכרון שהוקצה בצורה דינאמית, שני העצמים עלולים לשחרר את אותו הזיכרון.

תשובה לשאלה 26

כאשר הקומפיילר עובר על הקוד ומתרגם אותו לפקודות מכונה:

- התוכן של כל פונקציה מתורגם לפקודות שונות, אשר בתחילת ריצת התוכנית ייטענו לזיכרון.
- קריאה לפונקציה כלשהי מתורגמת לפקודת קפיצה לכתובת, שבה תימצא הפקודה הראשונה של הפונקציה.

כאשר מופיעה בקוד קריאה לפונקציה וירטואלית באמצעות מצביע לאובייקט כלשהו, הקומפיילר לא יודע איזה פונקציה בשרשרת ההורשה צריכה להתבצע, מכיוון שהסוג המדויק של האובייקט המוצבע נקבע רק בזמן ריצה. במקרה כזה, הקומפיילר לא יכול לתרגם את הקריאה לפונקציה לפקודת קפיצה לכתובת. הפתרון – V-Table:

- עבור כל מחלקה, הקומפיילר מייצר טבלה שבה רשומות כתובות הפונקציות הווירטואליות של המחלקה בזיכרון. טבלה כזאת נקראת V-Table, והיא הומצאה כדי לאפשר קריאה לפונקציות הווירטואליות האלה.
- כמו כן, הקומפיילר מוסיף לרשימת השדות של כל מחלקה מצביע לטבלת V-Table כלשהי.
- עבור כל קריאה לפונקציה וירטואלית באמצעות מצביע, הקומפיילר מקודד מספר פקודות מכונה, שבסופן תתבצע (בזמן ריצה) קפיצה לכתובת שרשומה ב-V-Table של האובייקט, שבאמצעותו נקראה הפונקציה.
- המצביע ל-V-Table שכל אובייקט מחזיק, מאותחל רק כאשר האובייקט נוצר (בזמן ריצה).

לפירוט נוסף – חלק 3 (חומר כללי: פונקציות מחלקה).

תשובה לשאלה 27

Singleton – תכן של מחלקה (Design Pattern), שמאפשר ליצור רק עצם אחד מסוגה בכל המערכת.
בקובץ ה-h נרשום:

```
class MyClass
{
public:
	static MyClass* CreateInstance(…);
private:
	MyClass(…);
	static MyClass* m_pInstance;
};
```

בקובץ ה-cpp נרשום:

```
MyClass* MyClass::m_pInstance = NULL;
MyClass* MyClass::CreateInstance(…)
{
	if (m_pInstance == NULL)
		m_pInstance = new MyClass(…);
	return m_pInstance;
}
```

הערות:

א. מכיוון שה-Constructor של המחלקה נמצא ב-private:
אי אפשר ליצור עצם מסוגה בצורה מפורשת, או לרשת אותה וליצור עצם של מחלקה שנגזרת ממנה.

ב. המשתנה הסטאטי של המחלקה יצביע לעצם היחיד מסוגה:
הפונקציה הסטאטית של המחלקה תיצור עצם חדש פעם אחת, ותחזיר מצביע אליו בכל פעם שתיקרא.

תשובה לשאלה 28

סמאפור זה מושג כללי, ו-Mutex זה סוג ספציפי של סמאפור:

- סמאפור הוא למעשה מנעול. ניתן להשתמש בו על מנת לסנכרן בין חוטים, להגביל את מספר המשתמשים במשאב כלשהו וכדומה. Mutex הוא סמאפור שהייעוד שלו הוא מניעה הדדית (Mutual Exclusion), על מנת להגן על קטע קוד קריטי. לדוגמא, אם חוט אחד כותב לקובץ כלשהו וחוט אחר קורא מאותו קובץ, יש לוודא שכל חוט מבצע את הקטע הקריטי שלו (רצף הפעולות שקשורות לקובץ הזה) מתחילתו ועד סופו, בלי שמתבצעת החלפת הקשר (Context Switch) בין החוטים תוך כדי הביצוע.
- לכל סמאפור מוגדר מספר כלשהו של חוטים שיכולים להשתמש בו לפני שהוא ננעל, וכל חוט נוסף שמנסה להשתמש בו יוצא להמתנה. Mutex הוא סמאפור שמספר החוטים שיכולים להשתמש בו לפני שהוא ננעל הוא אחד.
- שחרור סמאפור יכול להתבצע על ידי כל אחד מהחוטים שמכירים אותו. Mutex הוא סמאפור שרק החוט שנועל אותו יכול לשחרר אותו.

תשובה לשאלה 29

תקשורת בין תהליכים:

- תהליכים נוצרים ע"י מערכת ההפעלה, שמריצה כל אחד במרחב זיכרון משלו ובצורה בלתי תלויה באחרים.
- על מנת לקיים תקשורת בין תהליכים יש צורך במשאבים של מערכת ההפעלה, כמו Pipe או Message Box.

תקשורת בין חוטים:

- חוטים נוצרים ע"י תהליך במהלך הריצה שלו.
- התהליך הוא למעשה תוכנית שמורצת ע"י מערכת ההפעלה.
- בתחילת הריצה של התוכנית נוצר חוט אחד, שמייצג את השגרה הראשית של התוכנית.
- במהלך הריצה של התוכנית עשויים להיווצר חוטים נוספים, בהתאם למה שרשום בקוד התוכנית.
- מכיוון שכל החוטים שייכים לאותו התהליך (ולמעשה לאותה התוכנית), הם חולקים את מרחב הזיכרון שלו, ולכן התקשורת ביניהם יכולה להתבצע בצורה ישירה דרך מרחב הזיכרון הזה. בקוד התוכנית, אפשר לממש תקשורת בין חוטים באמצעות אובייקטים גלובאליים (משתנים, סמאפורים וכו'), אשר לכל החוטים של התוכנית יש גישה אליהם.

לפירוט נוסף – חלק 3 (חומר כללי: ניהול תהליכים וחוטים).

תשובה לשאלה 30

פונקציה שמקבלת שתי מחרוזות ומשווה ביניהן:

```
bool strcmp(const char* str1, const char* str2)
{
	int i;
	for (i=0; str1[i]!=0 && str2[i]!=0; i++)
		if (str1[i] != str2[i])
			return false;
	return str1[i] == str2[i];
}
```

תשובה לשאלה 31

ממשק של פונקציות עבור תור (FIFO):

```
void Create(Queue* pQueue);
void Destroy(Queue* pQueue);
bool IsFull(const Queue* pQueue);
bool IsEmpty(const Queue* pQueue);
bool Insert(Queue* pQueue, Item* pItem);
bool Extract(Queue* pQueue, Item** pItem);
```

תשובה לשאלה 33

Mutex זה סמאפור בינארי עם תכונה ייחודית, שנועדה לצורך הגנה על קטע קוד קריטי:

- שחרור סמאפור בינארי יכול להתבצע על ידי כל אחד מהחוטים שמכירים אותו.
- שחרור Mutex יכול להתבצע רק על ידי החוט שנועל אותו.

תשובה לשאלה 34

הבדלים בין תהליך (Process) לחוט (Thread):

א. תהליך לא יכול לגשת לשום חלק במרחב הזיכרון של תהליכים אחרים.
חוט יכול לגשת לזיכרון הגלובאלי של חוטים אחרים ששייכים לאותו תהליך.

ב. תקשורת בין תהליכים יכולה להתבצע רק בעזרת משאבים של מערכת ההפעלה.
תקשורת בין חוטים ששייכים לאותו תהליך יכולה להתבצע גם דרך הזיכרון הגלובאלי.

ג. הגנה על תהליך מפני פעולות של תהליכים אחרים היא באחריות מערכת ההפעלה בלבד.
הגנה על חוט מפני פעולות של חוטים אחרים ששייכים לאותו תהליך היא באחריות המתכנת.

ד. החלפת הקשר בין תהליכים כוללת את התוכן של מרחב הזיכרון של כל תהליך ואת ערכי הרגיסטרים.
החלפת הקשר בין חוטים ששייכים לאותו תהליך כוללת רק את המחסנית של כל חוט ואת ערכי הרגיסטרים.

לפירוט נוסף – חלק 3 (חומר כללי: ניהול תהליכים וחוטים).

תשובה לשאלה 35

פונקציה שמקבלת תו ובודקת אם הוא נמצא במערך של מיליון תווים:

```
bool check(char c)
{
	for (int i=0; i<N; i++) //Push i, Push N, Compare, Push i, Inc, Pop i
		if (arr[i] == c)  //Push arr, Push i, Add, Load, Push c, Compare
			return true;
	return false;
}
```

אפשר לחסוך את קריאת הערך של i פעם נוספת בכל איטרציה:

```
bool check(char c)
{
	char* end = arr+N;
	for (char* ptr=arr; ptr!=end; ptr++)
		if (*ptr == c)   //Push ptr, Load, Push c, Compare (removed "Push i, Add")
			return true;
	return false;
}
```

אפשר לחסוך את השוואת הערך של i עם מיליון בכל איטרציה:

```
bool check(char c)
{
	char x = arr[N-1];
	if (x == c)
		return true;
	arr[N-1] = c;
	for (int i=0; ; i++)	//Push i, Inc, Pop i (removed "Push i, Push N, Compare")
		if (arr[i] == c)
			break;
	arr[N-1] = x;
	return i<N-1;
}
```

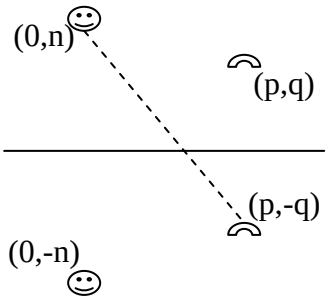
אפשר כמובן לשלב בין שתי האפשרויות.

תשובה לשאלה 36

נניח שהנהר, החוואי והפרה ממוקמים באופן הבא:

- גדת הנהר שקרובה לחוואי ולפרה – על ציר ה-X.
- החוואי – בנקודה (0,n), שנמצאת בחלק החיובי של ציר ה-Y.
- הפרה – בנקודה (p,q), שנמצאת ברבע החיובי של מערכת הצירים XY.

החוואי והפרה ממוקמים בנקודות קבועות. החוואי צריך ללכת אל הפרה דרך הנקודה (x,0), שנמצאת על גדת הנהר.
לכן, אפשר לתאר את המרחק שהחוואי צריך ללכת כפונקציה של הנקודה הנ"ל: $f(x) = \sqrt{(x^2+n^2)} + \sqrt{((p-x)^2+q^2)}$.
על מנת למצוא את המרחק המינימלי שהחוואי צריך ללכת, אפשר לגזור את הפונקציה ולהשוות את הנגזרת לאפס.
הבעיה היא שהנגזרת של הפונקציה היא די מסובכת, וקשה למצוא את הערך של x שבו היא מתאפסת.
במקום זה, נסתכל על ה"השתקפות" של החוואי והפרה ביחס לגדת הנהר שקרובה אליהם:



נעביר קו ישר בין החוואי וה"השתקפות" של הפרה.
נקודת החיתוך של הקו הזה עם גדת הנהר, היא הנקודה שאליה החוואי צריך ללכת.

<u>משוואת הישר:</u>

$n = a*0+b$, $-q = a*p+b$ ➔ $b = n$, $a = -(q+n)/p$ ➔ $y = -(q+n)/p*x+n$

<u>נקודת החיתוך:</u>

$y = -(q+n)/p*x+n$, $y = 0$ ➔ $-(q+n)/p*x+n = 0$ ➔ $x = n*p/(q+n)$ ➔ $[n*p/(q+n) , 0]$

תשובה לשאלה 37

נתונה צורה כאוסף של קודקודים. נחשב באופן חד-פעמי את המשוואות של כל אחת מהצלעות שלה.
בהינתן נקודה כלשהי, צריך לבדוק אם היא נמצאת בתוך הצורה, מחוץ לצורה או על השפה של הצורה:

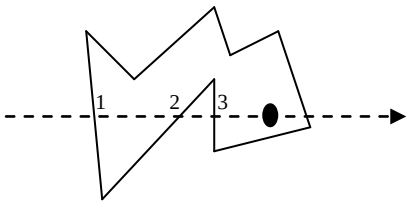
א. אם קיימת צלע, אשר נקודת הקלט מקיימת את המשוואה שלה וגם נמצאת בין שני הקודקודים שלה, אז נקודת הקלט נמצאת על השפה של הצורה. אחרת, היא נמצאת בתוך הצורה או מחוץ לצורה.

ב. נעביר דרך נקודת הקלט ישר שמקביל לציר ה-X, ונבחר נקודה על הישר שנמצאת מחוץ לצורה (נקודה שקואורדינטת ה-X שלה קטנה מקואורדינטות ה-X של כל קודקודי הצורה).

ג. נסרוק את הישר החל מאותה נקודה ועד לנקודת הקלט, ונספור כמה פעמים הוא חותך את צלעות הצורה (נתעלם מצלעות שנמצאות בדיוק על הישר).

ד. אם ספרנו מספר אי-זוגי של פעמים, אז נקודת הקלט נמצאת בתוך הצורה, ואם ספרנו מספר זוגי של פעמים, אז נקודת הקלט נמצאת מחוץ הצורה.

לדוגמא, בשרטוט הבא נקודת הקלט נמצאת בתוך הצורה, כי הישר שעובר דרכה חותך את הצורה שלוש פעמים:



תשובה לשאלה 38

רשימת הערכים שאיתה מאתחלים את המערך תהיה חלק מקוד התוכנית (קובץ ה-Image) בכל מקרה. אם המערך יוגדר בתוך הפונקציה, אז קוד הפונקציה יכיל (בנוסף לערכים) גם פקודות של השמת ערכים לתוך המערך. אם המערך יוגדר מחוץ לפונקציה, אז רשימת הערכים שלו תשב ב-Data Section ותיטען לזיכרון ביחד עם קוד התוכנית. כלומר, הקוד לא יכלול את פקודות ההשמה ולכן גודל התוכנית יהיה קטן יותר. בנוסף, הפונקציה לא תאתחל את המערך בכל פעם שהיא תיקרא, ולכן גם תרוץ מהר יותר.

תשובה לשאלה 39

נמלא מערך במספרים 0 עד N-1 (לפי הסדר), ונבצע N איטרציות.

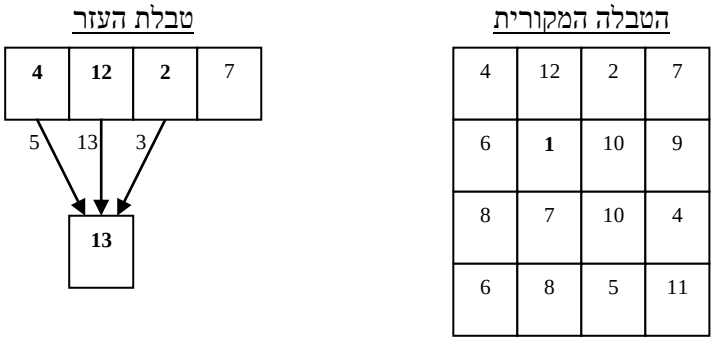
באיטרציה מספר i נחשב מספר אקראי j, ונחליף בין האיבר שיושב במקום ה-i לאיבר שיושב במקום ה-j:

```
for (i=0; i<N; i++)
        array[i] = i;
for (i=0; i<N; i++)
        Swap(array,i,Rand()%N);
```

תשובה לשאלה 40

נשתמש בטבלת עזר (תכנון דינאמי) – בכל משבצת בטבלת העזר נרשום את הסכום הכי גדול של מסלול כלשהו שמסתיים במשבצת שמקבילה לה בטבלה המקורית:

1. נתחיל בשורה הראשונה, ובכל משבצת פשוט נרשום את הערך שרשום במשבצת שמקבילה לה.
2. בכל שורה לאחר מכן, לכל משבצת נחשב את הסכומים האפשריים עבורה (בעזרת הערך שרשום במשבצת שמקבילה לה והסכומים שחישבנו בשורה הקודמת), ונרשום בה את הסכום הגדול ביותר מביניהם. לדוגמא, עבור המשבצת השניה בשורה השניה נחשב את הערך המתאים באופן הבא:



3. לסיום, נעבור על השורה האחרונה בטבלת העזר ונמצא את הסכום הכי גדול שרשום בה.

סיבוכיות זמן – O(N^2), סיבוכיות זיכרון – O(N^2).

תשובה לשאלה 42

נבנה גרף באופן הבא:

- עבור כל זמר a, נגדיר צומת N(a).
- עבור כל יחס b<c, נגדיר קשת מ-N(b) ל-N(c).

בהינתן שני זמרים x ו-y:

- נסרוק את הגרף החל מ-N(x). אם במהלך הסריקה נגיע ל-N(y), אז נוכל לקבוע ש-y יותר טוב מ-x.
- אחרת, נסרוק את הגרף החל מ-N(y). אם במהלך הסריקה נגיע ל-N(x), אז נוכל לקבוע ש-x יותר טוב מ-y.
- אחרת, נוכל להכריז שלא ניתן לדעת מי יותר טוב מביניהם.

תשובה לשאלה 43

המשימה: צריך לממש פונקציה שמקצה 8 בתים כל עוד יש מקום פנוי בזיכרון, ופונקציה שמשחררת 8 בתים.
הבעיה: צריך לעקוב אחרי המקומות הפנויים בזיכרון. בהתחלה כולו פנוי, אבל בהמשך יכולים להיווצר "חורים".
שיטת העבודה:

- ננהל את הזיכרון הפנוי כ"רשימה מקושרת". בכל שטח פנוי, נרשום את הכתובת של השטח הפנוי הבא.
  מכיוון שגודל הזיכרון הוא 1MB, כל כתובת היא באורך של 20 ביטים, ולכן ניתנת לשמירה בשטח של 8 בתים.
- על מנת להקצות שטח, "נוריד" את השטח הראשון ברשימה. על מנת לשחרר שטח, "נוסיף" אותו לתחילת הרשימה.

מבנה הנתונים:

- נחזיק משתנה גלובאלי dword* listHead, שיצביע לתחילת הרשימה.
- את הרשימה נאתחל באופן חד-פעמי, לאחר עליית המערכת (בכל שטח נרשום את הכתובת של השטח הבא).

פונקציה להקצאת הזיכרון:

```
void* malloc()
{
	void* p = (void*)listHead;
	listHead = (dword*)(*listHead);
	return p;
}
```

פונקציה לשחרור הזיכרון:

```
void free(void* p)
{
	*((dword*)p) = listHead;
	listHead = (dword*)p;
}
```

תשובה לשאלה 44

כאשר תוכנית מכילה משתנים או אובייקטים גלובאליים, האתחול שלהם חייב להתבצע לפני שהפונקציה main נקראת.
אתחול של משתנה גלובאלי בערך ידוע מראש (לדוגמא, int x=5), כלל לא מתורגם לקוד. ערך כזה "מוטמע" בתוכנית (כחלק מקובץ ה-Image) ונטען איתה לזיכרון כאשר היא מתחילה לרוץ. אתחול של משתנה גלובאלי בערך שאינו ידוע מראש (לדוגמא, int z=x+y), מתורגם לקוד שמתבצע אחרי שהתוכנית נטענת לזיכרון ולפני שהפונקציה main נקראת.
גם הצהרה על אובייקט גלובאלי אשר מוגדר עבורו Constructor, מתורגמת לקוד שמתבצע כבר בשלב הזה.
לסיכום, תוכנית שנכתבה ב-C או ב-++C, לא בהכרח מתחילה לרוץ מהפונקציה main.

תשובה לשאלה 45

הבדלים שחשוב להתייחס אליהם במהלך כתיבת קוד גנרי:

- הדרך שבה המעבד מתרגם יחידות זיכרון (כמו word או dword):
  - מעבדי Little Endian מתייחסים לבית הראשון ביחידה כ-LSB ולבית האחרון ביחידה כ-MSB.
  - מעבדי Big Endian מתייחסים לבית הראשון ביחידה כ-MSB ולבית האחרון ביחידה כ-LSB.
- הדרך שבה הקומפיילר מתרגם מבני Bits-Fields (כמו struct Byte{uint8 b1:1,b2:1,…b8:1}):
  - ישנם קומפיילרים שמתייחסים לביט הראשון במבנה כ-LSB ולביט האחרון במבנה כ-MSB.
  - ישנם קומפיילרים שמתייחסים לביט הראשון במבנה כ-MSB ולביט האחרון במבנה כ-LSB.

שני ההבדלים האלה יכולים להיות משמעותיים בעיקר כאשר מדובר בקידוד ופענוח של נתונים שעוברים בתקשורת.
נקודה נוספת שחשוב להתייחס אליה, היא הגודל שהקומפיילר מקצה לטיפוסים שונים (כמו char, short, int, long).

תשובה לשאלה 46

מימוש הפונקציה:

```
void memcopy(void* dst, void* src, int len)
{
	for (int i=0; i<len; i++)
		((byte*)dst)[i] = ((byte*)src)[i];
}
```

אפשר לשפר את זמן הריצה של הפונקציה בכמה דרכים:

א. שימוש בטיפוס יותר גדול מ-byte (לדוגמא, word או dword), על מנת לבצע מספר קטן יותר של איטרציות. דבר כזה יחייב שמספר הבתים (len) יתחלק בשלמות בגודל של הטיפוס, וגם שהמצביעים (src ו-dst) יהיו Aligned לגודל של הטיפוס, או שהמעבד יוכל לבצע Unaligned Load/Store במידה והם לא.

ב. קידום של המצביעים src ו-dst במקום שימוש באינדקס i, על מנת לחסוך את קריאת הערך שלו בכל איטרציה. כלומר, רצף הפקודות [Push ptr, Push i, Add, Load] יוחלף ברצף הפקודות [Push ptr, Load, Load], עבור כל אחד מהמצביעים הנ"ל.

מצבים שבהם הפונקציה לא תעתיק את הנתונים במדויק, יכולים להתרחש כאשר:

- חוט אחר רץ במקביל, ומשנה את התוכן של אחד מקטעי הזיכרון (כאשר הגישה אליהם היא לא בלעדית).
- המשתנים src ו-dst לא מצביעים לקטעי זיכרון באורך של len בתים (שהוקצו ע"י התוכנית בשלב כלשהו).
- קטעי הזיכרון src[0…len-1] ו-dst[0…len-1] הם לא בלתי תלויים (כלומר, חולקים שטח משותף כלשהו).

תשובה לשאלה 47

Context Switch בין Task'ים לא יכול להתבצע כאשר הפסיקות מנוטרלות.

תשובה לשאלה 48

כאשר בפונקציה רשום a=b, לא תמיד נוכל לדעת בוודאות מהו הערך של a, מכיוון שלא ידוע איך הפונקציה נקראה (ע"י איזה Task ועם איזה פרמטרים). על מנת להבטיח שתמיד נוכל לדעת בוודאות מהו הערך של a, נצטרך להתייחס לקטע הקוד הזה (a=b) כאל קטע קוד קריטי, ולהגן עליו בהתאם. ישנן מספר אפשרויות:

א. ניטרול ואיפשור הפסיקות (Disable/Enable Interrupts).

ב. המתנה על משתנה גלובאלי אחר (Busy Wait).

ג. שימוש בסמאפור ייעודי (Mutex).

כאשר בפונקציה רשום b=a: אותה תשובה, אלא אם כן הערך של a נשאר קבוע במשך כל התוכנית.

לפירוט נוסף – חלק 3 (חומר כללי: ניהול תהליכים וחוטים).

תשובה לשאלה 49

בפונקציה הנתונה בשאלה קיימות שתי בעיות:

א. הפונקציה מחזירה את הכתובת שבה נמצא המערך buff. מכיוון שהמערך הזה מוקצה על המחסנית של הפונקציה (הקצאה סטאטית), הוא "מפסיק להתקיים" כאשר הפונקציה מסתיימת, ולכן הכתובת שתוחזר לא תכיל ערכים מוגדרים (אלא "זבל").

ב. הפונקציה מעתיקה לתוך המערך buff את המחרוזות שמוצבעות ע"י first ו-last. אם סכום האורכים של המחרוזות האלה הוא יותר גדול מהאורך של buff, אז הפעולה הזאת תגרום לדריסת זיכרון.

אפשר לפתור את שתי הבעיות ע"י הקצאה דינאמית: `char* buff = new char[strlen(first)+strlen(last)+1]` .

תשובה לשאלה 50

התהליך של קפיצה לשגרת פסיקה וחזרה ממנה, מתבצע באופן דומה לתהליך של קפיצה לפונקציה וחזרה ממנה:

- לפני קפיצה לשגרת פסיקה או לפונקציה, הערך של רגיסטר ה-Instruction Pointer נדחף למחסנית.
- בסיום שגרת פסיקה או פונקציה, הערך של רגיסטר ה-Instruction Pointer נשלף מהמחסנית.

בדרך כלל, פקודת החזרה משגרת פסיקה נקראת RetI ופקודת החזרה מפונקציה נקראת RetF.

תשובה לשאלה 51

על מנת לנתק צומת מרשימה מקושרת, צריך לחבר את הצומת שקודם לו לצומת שבא אחריו.
על מנת לבצע את זה ב-O(1), מספיק לדעת מהו הצומת שקודם לו, וממנו אפשר להגיע ב-O(1) גם לצומת שבא אחריו.
מכיוון שהרשימה המקושרת היא חד-כיוונית ולא ניתן למצוא את הצומת הקודם ב-O(1), נעביר את הבעיה לצומת הבא.
נשמור באופן זמני את הנתונים של הצומת הבא, נמחוק אותו ונעתיק אותם אל הצומת הנוכחי.

```
void Cut(Node* curr)
{
	Node temp = *(curr->next);
	free(curr->next);
	*curr = temp;
}
```

הסבר:
הפונקציה לא מוחקת "פיזית" את הצומת המבוקש, אלא מעבירה אליו את התוכן של הצומת הבא אחריו (שאותו הפונקציה כן מוחקת "פיזית"). התוצאה הסופית היא רשימה מקושרת, ש"נראית" כאילו הצומת המבוקש נמחק ממנה.

תשובה לשאלה 52

נתונה רשימה מקושרת חד-כיוונית. נבדוק אם יש בה מעגל, באופן הבא:

1. נקצה צומת נוסף (מנותק מהרשימה).
2. נעבור על הרשימה, ועבור כל צומת נשנה את הצומת שהצביע אליו, כך שיצביע אל הצומת הנוסף.
3. אם בשלב כלשהו נגיע לצומת הנוסף, נדע שיש מעגל. אם בשלב כלשהו נגיע ל-NULL, נדע שאין מעגל.

```
bool SearchLoop(Node* head)
{
	Node temp;
	while (head!=&temp && head!=NULL)
	{
		Node* next = head->next;
		head->next = &temp;
		head = next;
	}
	return head == &temp;
}
```

הערות:

א. סיבוכיות הזמן של האלגוריתם היא O(N).
ב. פירוט של שיטות אחרות לפתרון הבעיה, אפשר למצוא בתשובה לשאלה 84.

תשובה לשאלה 55

התוכנית הנתונה בשאלה תדפיס 42, מכיוון שהמשתנה x שהפונקציה func מקבלת הוא משתנה מקומי, ולכן שינוי הערך שלו לא ישפיע על הערך של המשתנה שנשלח אליה מהפונקציה main (הכתובת של המשתנה y).
הבעיה בתוכנית, היא שהפונקציה func מקצה זיכרון וכותבת בו (באמצעות המשתנה המקומי x) את הערך 17. כאשר היא מסתיימת, המצביע "נאבד" ולא ניתן לשחרר את הזיכרון שהוקצה (התוצאה – דליפת זיכרון).

תשובה לשאלה 56

הפונקציה הנתונה בשאלה, מבצעת את הפעולות הבאות:

- בהתחלה, היא מקצה זיכרון שמוצבע ע"י המשתנה ptr.
- במהלך הלולאה, היא מגדילה את הערך של המשתנה ptr.
- בסיום, היא משחררת את הזיכרון שמוצבע ע"י המשתנה ptr.

כלומר, היא מקצה זיכרון במקום אחד ומשחררת זיכרון ממקום אחר.

תשובה לשאלה 57

מכיוון שהמשתנה a הוא משתנה מקומי בפונקציה func, הוא יתקיים בנפרד בכל Thread:
ב-Thread הראשון הערך שלו יהיה 1, וב-Thread השני הערך שלו יהיה 2.

תשובה לשאלה 58

אפשרות ראשונה:

- נוציא גרב אחת, ובארגז יישארו 29 גרביים. גרב אחת מתוכן היא באותו צבע, וההסתברות להוציא אותה היא 1/29.
- הפעולה הזאת שקולה להוצאה של שתי גרביים, ולכן ההסתברות להוציא שתי גרביים מאותו צבע היא גם 1/29.

אפשרות שניה:

- מספר האפשרויות השונות לבחור 2 גרביים מתוך 30, הוא "30 מעל 2" (!30/!2/!28), כלומר 15*29.
- מספר זוגות הגרביים הוא 15, ולכן ההסתברות להוציא זוג גרביים היא 15/(15*29), כלומר 1/29.

תשובה לשאלה 59

שיטות לבדוק מהי "צריכת ה-Stack" המקסימלית:

א. אפשר למלא את ה-Stack בערך קבוע כלשהו, להריץ את התוכנית הרבה פעמים ולמצוא את גודל השטח שכבר לא מכיל את הערך הזה. לדוגמא: for (int i=0; i<STACK_SIZE; i++) SP[i]=0x12345678 .

ב. אפשר לשמור את הערך המקסימלי של ה-Stack Pointer במספר מקומות בתוכנית, להריץ אותה הרבה פעמים ולמצוא את הערך המקסימלי מבין כל אלה שהתקבלו. לדוגמא: if (MaxSP<SP) MaxSP=SP .

תשובה לשאלה 61

כאשר פונקצית מחלקה מבצעת delete this, מערכת ההפעלה מנסה לשחרר את שטח הזיכרון שבו נמצא האובייקט, שבאמצעותו הפונקציה נקראה. אם שטח הזיכרון הזה לא הוקצה בצורה דינאמית, אז סביר להניח שהתוכנית תקרוס. כלומר, למעשה ניתן לקרוא לפונקציה כזאת רק באמצעות אובייקט שהוקצה בצורה דינאמית.

תשובה לשאלה 62

העברת משתנה By Reference היא מעין "הכלאה" של העברת משתנה By Value והעברת משתנה By Pointer:

- בזמן כתיבת התוכנית הוא "נראה" כמו משתנה שמועבר By Value (הצורה שבה פונים אליו בקוד).
- במהלך ריצת התוכנית הוא "מתנהג" כמו משתנה שמועבר By Pointer (הצורה שבה הוא נשלח במחסנית).

יש לשיטה הזאת מספר יתרונות:

א. ביצוע Casting בצורה "בטוחה" – כאשר משתנה מועבר כמצביע, ניתן לבצע עליו Casting למצביע מכל סוג אחר, בלי שיתקבלו שגיאות או אזהרות בזמן קומפילציה. כתוצאה מכך, עלולות להיות שגיאות בזמן ריצה.

ב. הגדרת Copy Constructor – אפשרית רק כאשר הוא מקבל את האובייקט By Reference. העברת האובייקט By Value תהיה שקולה לקריאה רקורסיבית ל-Copy Constructor (כי האובייקט יווצר מחדש במחסנית).

ג. יצירת מחלקות Template – פניה לאובייקט שמועבר By Reference נכתבת בקוד כמו פניה לאובייקט שמועבר By Value. הדבר הזה מאפשר לכתוב קוד אחיד וליצור Template'ים עבור שני הסוגים (בלי "כוכבית" או "חץ").

תשובה לשאלה 63

על מנת שאובייקט מסוג D יכיל אובייקט אחד מסוג A, צריך להגדיר את מחלקות B ו-C באופן הבא:

- class B : virtual public A
- class C : virtual public A

על מנת שאובייקט מסוג D יכיל שני אובייקטים מסוג A, צריך להגדיר את מחלקות B ו-C באופן הבא:

- class B : public A
- class C : public A

כאשר אובייקט מסוג D מכיל שני אובייקטים מסוג A, הגישה לכל אחד מהם צריכה להתבצע באופן הבא:

- B::var , B::func()
- C::var , C::func()

בדוגמא הנ"ל, var ו-func הם משתנה ופונקציה של מחלקה A, והגישה אליהם מתבצעת דרך מחלקות B ו-C.

תשובה לשאלה 64

פונקציה שהופכת מטריצה של מספרים, ללא שימוש בזיכרון נוסף:

```
void Transpose(int** matrix, int n)
{
	for (int i=0; i<n; i++)
	{
		for (int j=i; j<n; j++)
		{
			int temp = matrix[i][j];
			matrix[i][j] = matrix[j][i];
			matrix[j][i] = temp;
		}
	}
}
```

תשובה לשאלה 65

תקלה מתרחשת במצב הבא:

- תחנה A שולחת Msg כלשהו, ואף אחד מ-1000 ה-Ack'ים שתחנה H מחזירה לאחר מכן, לא מתאים לו.

על מנת לזהות תקלה, תחנה B צריכה:

- לשמור תור FIFO שמכיל את 1000 ה-Msg'ים האחרונים שתחנה A שלחה.
- לסמן כל Msg שנמצא בתור הזה, ברגע שתחנה H מחזירה Ack שמתאים לו.

בכל פעם שתחנה A שולחת Msg חדש, תחנה B צריכה:

- להוציא את ה-Msg הראשון בתור.
- להודיע על תקלה אם הוא לא מסומן.
- להכניס את ה-Msg החדש לסוף תור.

בכל פעם שתחנה H מחזירה Ack חדש, תחנה B צריכה:

- למצוא בתור את ה-Msg המתאים ל-Ack החדש.
- לסמן את ה-Msg הזה כאחד שהתקבל Ack עבורו.

על מנת שמציאת Msg תוכל להתבצע ב-O(1), תחנה B צריכה:

- להחזיק Hash Table שתאפשר למצוא כל Msg בתור ב-O(1).

הערה: תחנה B צריכה להתייחס ל-Msg'ים ול-Ack'ים רק באמצעות המספרים המזהים שלהם.

תשובה לשאלה 66

הבעיה במצב המתואר בשאלה, היא ש-A עלול להמתין ש-C יסיים להשתמש במשאב המשותף, בזמן ש-B מונע מ-C לרוץ כי הוא בעדיפות גבוהה יותר. כלומר, B עלול בצורה עקיפה למנוע מ-A לרוץ, למרות שהוא בעדיפות נמוכה יותר.

הפתרון הוא לבצע Priority Inversion:

לפני ש-C ניגש אל המשאב המשותף, הוא צריך להעלות את העדיפות שלו, כך שהיא תהיה מעל העדיפות של B ומתחת לעדיפות של A. כאשר C מסיים להשתמש במשאב המשותף, הוא צריך להחזיר את העדיפות שלו לערך המקורי שלה.

תשובה לשאלה 67

מציאת המיקום של הביט השלישי שערכו 1, ברגיסטר של 32 ביטים:

```
int FindThird1(int reg)
{
	int count = 0;
	for (int i=0; i<32; i++)
	{
		count += (reg>>i)&1;
		if (count == 3)
			return i;
	}
	return -1;
}
```

שיפור זמן הריצה של הפונקציה – נחשב באופן חד-פעמי את המערכים הבאים:

- NumOf1s[256] – עבור כל ערך אפשרי ליחידה של 8 ביטים, נרשום את מספר הביטים ביחידה שערכם 1.
- First1[256] – עבור כל ערך אפשרי ליחידה של 8 ביטים, נרשום את מיקום הביט הראשון ביחידה שערכו 1.
- Second1[256] – עבור כל ערך אפשרי ליחידה של 8 ביטים, נרשום את מיקום הביט השני ביחידה שערכו 1.
- Third1[256] – עבור כל ערך אפשרי ליחידה של 8 ביטים, נרשום את מיקום הביט השלישי ביחידה שערכו 1.

מציאת המיקום של הביט השלישי שערכו 1 (בעזרת המערכים הנ"ל):

```
int FindThird1Fast(int reg)
{
	int count = 0;
	int* Locations[4] = {NULL, First1, Second1, Third1};
	for (int n=0; n<4; n++)
	{
		int unit = (byte)(reg>>(8*n));
		if (count + NumOf1s[unit] < 3)
			count += NumOf1s[unit];
		else
			return 8*n+Locations[3-count][unit];
	}
	return -1;
}
```

הסבר:

- עוברים על כל בית ברגיסטר, וסוכמים (בעזרת המערך הראשון) את מספר הביטים שערכם 1.
- כאשר מגיעים ל-3 או יותר, מוצאים (בעזרת שאר המערכים) את המיקום של הביט השלישי שערכו 1.

תשובה לשאלה 68

נכין רשימה ממוינת לפי זמני ההתחלה של ה-Task'ים, ורשימה ממוינת לפי זמני הסיום של ה-Task'ים.
נעדכן מונה שיספור את מספר Task'ים שרצו בכל רגע נתון, ומשתנה שישמור את הערך המקסימלי של המונה הזה.
נעבור על הרשימה הראשונה, ובאיטרציה מספר X:

1. נוסיף למונה ב-1.
2. נמצא ברשימה השניה את ה-Task הראשון שזמן הסיום שלו גדול מזמן ההתחלה של Task מספר X.
3. נחסיר מהמונה את מספר ה-Task'ים שהסתיימו לפני ש-Task מספר X התחיל ואחרי ש-Task מספר X-1 התחיל.

סיבוכיות זמן (כאשר מספר ה-Task'ים במערכת הוא N):

- את סעיף 2 ניתן לבצע בעזרת חיפוש בינארי, כלומר ב-O(log(N)).
- את שני הסעיפים האחרים ניתן לבצע ב-O(1).
- סה"כ – O(N*log(N)).

תשובה לשאלה 71

החבר צריך לשלוח לך מנעול קפיצי פתוח.
אתה צריך לשים את המכתב בתוך מזוודה, לנעול אותה עם המנעול הקפיצי ולשלוח אליו חזרה.

תשובה לשאלה 72

נסמן:

- S – מרחק הנסיעה כולה.
- V – מהירות הנסיעה הממוצעת.
- T_1 – זמן הנסיעה בכיוון הלוך.
- T_2 – זמן הנסיעה בכיוון חזור.

ידוע לנו:

- $T_1 = ½S / 40$ (נתון)
- $V = S / (T_1+T_2)$ (מהירות שווה מרחק חלקי זמן)

ולכן נובע:

- $T_2 = S / V - ½S / 40$

אם נרצה שיתקיים:

- $V = 80$

אז נצטרך שיתקיים:

- $T_2 = S / 80 - ½S / 40 = 0$

וזה בלתי אפשרי (לנסוע חזרה ב"אפס זמן").

תשובה לשאלה 73

נניח שהמספר הראשון נתון ברגיסטר A והמספר השני נתון ברגיסטר B. נבצע את הפעולות הבאות לפי הסדר:

אפשרות ראשונה	אפשרות שניה
• A = A + B	• A = A ^ B
• B = A – B	• B = A ^ B
• A = A – B	• A = A ^ B

הערה: באפשרות הראשונה תיתכן גלישה במהלך החישובים, אבל היא לא תשפיע על התוצאה הסופית.

תשובה לשאלה 74

אלגוריתם לסידור המשתנים בסדר עולה:

1. נבנה לפי הטבלה גרף, שבו כל צומת מייצג משתנה כלשהו, ולכל שני צמתים שמייצגים X ו-Y, אם X<Y אז יש קשת מ-X ל-Y. כמובן שלא יכולים להיות מעגלים בגרף, כי לא יכול להיות מצב שבו X<…<Z<Y<X. בהנחה שלכל זוג משתנים נתון בטבלה היחס ביניהם, נקבל גרף של רכיב אחד (ולא של מספר רכיבים בלתי קשירים).
2. נבצע על הגרף מיון טופולוגי באופן הבא:
   א. נקצה רשימה שבה נרשום את המשתנים בסדר עולה, נבחר צומת כלשהו ונבצע חיפוש לעומק (DFS).
   ב. כל פעם ש"נסיים" צומת (לאחר שעברנו על כל הקשתות שיוצאות ממנו), נכניס אותו לתחילת הרשימה.

הערה: המיון הטופולוגי אפשרי, מכיוון שאין מעגלים בגרף.

תשובה לשאלה 75

האסירים מחלקים לעצמם מספרים בין 1 ל-N, וקובעים ביניהם מה כל אסיר צריך לעשות כאשר הוא יוצא החוצה. בנוסף, לפני שהמנהל מחזיר אותם לתאים, הם דואגים שהמנורה תהיה כבויה.

- כל אסיר שמספרו קטן מ-N, מדליק את המנורה אם היא כבויה ובתנאי שהוא עדיין לא הדליק אותה אף פעם.
- האסיר שמספרו הוא N, מכבה את המנורה אם היא דלוקה וסופר את מספר הפעמים שהוא כיבה אותה.
- לאחר שספר N-1 פעמים, הוא מודיע למנהל שכל אסירים כבר יצאו לחצר לפחות פעם אחת.

תשובה לשאלה 76

אפשר לגלות מי הם שני המספרים החסרים, גם אם אחד מהם הוא המינימום או אחד מהם הוא המקסימום (או גם וגם).
בנוסף, אפשר לעשות את זה במעבר אחד על המערך (אז אולי חסרים תנאים מגבילים בשאלה הזאת):

1. נעבור על המערך ונחשב את סכום המספרים שלו ואת מכפלת המספרים שלו (נסמן S_0 ו-P_0 בהתאמה).
2. מכיוון ש-m ו-n ידועים, ניתן לחשב את הסכום ואת המכפלה של <u>כל המספרים</u> בתחום $[m,\ldots,m+n+1]$.
3. נחשב את סכום שני המספרים שחסרים ואת מכפלת שני המספרים שחסרים (נסמן S ו-P בהתאמה):
   $S = \sum[m,\ldots,m+n+1] - S_0$, $P = \prod[m,\ldots,m+n+1] - P_0$.
4. נסמן את שני המספרים שחסרים x_1,x_2. מתקיים: $x_1 + x_2 = S$ וגם $x_1 * x_2 = P$.
5. קיבלנו שתי משוואות בשני נעלמים, ומכאן הפתרון הוא פשוט: $x_1,x_2 = [-S \pm \sqrt{(S^2-4P)}] / -2$.

הערה: עברנו על המערך בסך הכל פעם אחת בלבד (בשלב מספר 1).

תשובה לשאלה 77

מכיוון שיש 4 מתגים שכל אחד מהם נמצא באחד מ-2 מצבים (On או Off), ישנם סה"כ 16 מצבים אפשריים לשולחן כולו, כאשר ב-2 מתוכם הנורה דולקת. נשייך את 2 המצבים האלה לקבוצה אחת, ונחלק את 14 המצבים הנותרים לקבוצות של מצבים שהם סימטריים זה לזה (ביחס לשולחן):

A	0101, 1010	שני מתגים נגדיים ב-On ושני מתגים נגדיים ב-Off
B	0011, 0110, 1100, 1001	שני מתגים סמוכים ב-On ושני מתגים סמוכים ב-Off
C	0001, 0010, 0100, 1000	מתג אחד ב-On ושלושה מתגים ב-Off
D	0111, 1110, 1101, 1011	שלושה מתגים ב-On ומתג אחד ב-Off
N	0000, 1111	נורה דולקת

המטרה היא למצוא רצף של פעולות, שיביא כל אחד מהמצבים בקבוצות A, B, C ו-D למצב כלשהו בקבוצה N:

- עבור קבוצה A קיימת פעולה X, שמעבירה כל אחד מהמצבים בה למצב כלשהו בקבוצה N.
- לכן, מספיק למצוא רצף של פעולות, שיביא כל אחד מהמצבים בקבוצות B, C ו-D למצב כלשהו בקבוצות A או N.
- עבור קבוצה B קיימת פעולה Y, שמעבירה כל אחד מהמצבים בה למצב כלשהו בקבוצות A ו-N.
- לכן, מספיק למצוא רצף של פעולות, שיביא כל אחד מהמצבים בקבוצות C ו-D למצב כלשהו בקבוצות A, B או N.
- עבור קבוצות C ו-D קיימת פעולה Z, שמעבירה כל אחד מהמצבים בהן למצב כלשהו בקבוצות A, B או N.

נגדיר את הפעולות הנ"ל, אשר מתוכן נבצע אחת בכל שלב (לאחר שהמפסק הראשי יורד):

X	^ 0101	לחיצה על המתג הראשון ועל המתג השלישי
Y	^ 0011	לחיצה על המתג הראשון ועל המתג השני
Z	^ 0001	לחיצה על המתג הראשון

נגדיר את סדר הפעולות (משמאל לימין) אשר במהלכו או בסיומו הנורה תדלוק, בלי קשר למצב ההתחלתי של השולחן:

פעולה / מצב התחלתי	X	Y	X	Z	X	Y	X
A	N						
B	B	A, N	N				
C	C, D	C, D	C, D	A, B, N	B, N	A, N	N
D	C, D	C, D	C, D	A, B, N	B, N	A, N	N

תשובה לשאלה 78

נתייחס לסודוקו כאל טבלה של 9 שורות, שבכל אחת ישנן 9 משבצות.
בכל שורה נתחיל במשבצת אחרת, ונמלא את המספרים 1 עד 9 לפי הסדר:

- בשורה מספר 1 נתחיל במשבצת מספר 1.
- בשורה מספר 2 נתחיל במשבצת מספר 4.
- בשורה מספר 3 נתחיל במשבצת מספר 7.
- בשורה מספר 4 נתחיל במשבצת מספר 2.
- בשורה מספר 5 נתחיל במשבצת מספר 5.
- בשורה מספר 6 נתחיל במשבצת מספר 8.
- בשורה מספר 7 נתחיל במשבצת מספר 3.
- בשורה מספר 8 נתחיל במשבצת מספר 6.
- בשורה מספר 9 נתחיל במשבצת מספר 9.

1	2	3	4	5	6	7	8	9
7	8	9	**1**	2	3	4	5	6
4	5	6	7	8	9	**1**	2	3
9	**1**	2	3	4	5	6	7	8
6	7	8	9	**1**	2	3	4	5
3	4	5	6	7	8	9	**1**	2
8	9	**1**	2	3	4	5	6	7
5	6	7	8	9	**1**	2	3	4
2	3	4	5	6	7	8	9	**1**

הנוסחא הכללית: Table[i][j] = ((i * 6 - i / 3 + j) % 9) + 1
סימון: Table – טבלה של 9 × 9 משבצות, i,j – אינדקסים בין 0 ל-8

תשובה לשאלה 79

קיימים שלושה סוגים של סוגריים: ( ), [ ], { }.
נחלק את הסוגים השונים לשתי קבוצות:

א. סוגרים שמאליים הם כאלה שמתחילים סוגריים: (, [, {.
ב. סוגרים ימניים הם כאלה שמסיימים סוגריים:),], }.

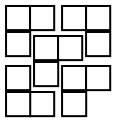
על מנת לבדוק אם ביטוי שמכיל את שלושת הסוגים הוא חוקי:

- נסרוק אותו משמאל לימין באופן הבא:
  - כאשר ניתקל בסוגר שמאלי, נכניס אותו למחסנית (פעולת Push).
  - כאשר ניתקל בסוגר ימני, נבדוק אם המחסנית ריקה. אם כן, נפסיק את סריקת הביטוי ונכריז שהוא לא חוקי. אחרת, נוציא את האיבר האחרון שהכנסנו למחסנית (פעולת Pop) ונבדוק אם הוא סוגר שמאלי מאותו סוג. אם לא, נפסיק את סריקת הביטוי ונכריז שהוא לא חוקי.
- במידה וסיימנו את סריקת הביטוי עד סופו, נבדוק אם המחסנית ריקה. אם לא, נכריז שהוא לא חוקי. אחרת, נדע שהוא חוקי.

אם קיים סוג רביעי של סוגריים, שבו אין הבדל בין סוגר שמאלי לסוגר ימני: | |, אז כאשר ניתקל בסוגר מהסוג הזה, לא נוכל לדעת אם לבצע Push או Pop. לכן, נוציא את האיבר האחרון שהכנסנו למחסנית ונבדוק אם הוא מאותו סוג. אם לא, נכניס אותו בחזרה למחסנית, ובנוסף נכניס גם את הסוגר שניתקלנו בו למחסנית.

תשובה לשאלה 80

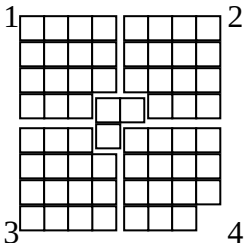
עבור N=1, נקבל שטח בגודל 2×2 ללא המשבצת הימנית התחתונה, שאותו נוכל לרצף באמצעות הצורה 

עבור N=2, נקבל שטח בגודל 4×4 ללא המשבצת הימנית התחתונה, שאותו נוכל לרצף באמצעות הצורות 

נניח שניתן לרצף שטח בגודל 2^N × 2^N ללא המשבצת הימנית התחתונה, עבור N=k.
נוכיח שניתן לרצף שטח בגודל 2^N × 2^N ללא המשבצת הימנית התחתונה, עבור N=k+1:

- לפי ההנחה, ניתן לרצף שטח בגודל 2^k × 2^k ללא המשבצת הימנית התחתונה.
- ניקח את השטח הזה, נשכפל אותו ארבע פעמים, ונמקם כל חלק באופן הבא:
  - את חלק מספר 1 נמקם בפינה השמאלית העליונה, כך שהמשבצת החסרה תהיה בפינה הימנית התחתונה שלו.
  - את חלק מספר 2 נמקם בפינה הימנית העליונה, כך שהמשבצת החסרה תהיה בפינה השמאלית התחתונה שלו.
  - את חלק מספר 3 נמקם בפינה השמאלית התחתונה, כך שהמשבצת החסרה תהיה בפינה הימנית העליונה שלו.
  - את חלק מספר 4 נמקם בפינה הימנית התחתונה, כך שהמשבצת החסרה תהיה בפינה הימנית התחתונה שלו.
- את המרווח שנוצר בין המשבצות החסרות של שלושת החלקים הראשונים, נוכל לרצף באמצעות הצורה 
- ביחד עם החלק הרביעי, נקבל שטח בגודל 2^(k+1) × 2^(k+1) ללא המשבצת הימנית התחתונה.

לדוגמא, באמצעות השטח שקיבלנו עבור N=2, נבנה שטח עבור N=3:



מספר הצורות שבהן נשתמש בשביל לרצף שטח בגודל 2^n × 2^n ללא המשבצת הימנית התחתונה:

- $A_1 = 1$
- $A_{n+1} = 4A_n + 1$

תשובה לשאלה 81

תוכנית שבודקת כמה ביטים דלוקים יש בבית אחד:

```
int CountOnes(byte x)
{
	#define BITS_PER_BYTE (8*sizeof(byte))
	int res = 0;
	for (int i=0; i<BITS_PER_BYTE; i++)
		res += (x>>i)&1;
	return res;
}
```

על מנת שהתוכנית תתבצע בזמן קבוע, אפשר להיעזר בטבלה (Hash Table):

```
int CountOnesFast(byte x)
{
	#define VALS_PER_BYTE (1<<BITS_PER_BYTE)
	static bool first_time = true;
	static int table[VALS_PER_BYTE] = {0};
	if (first_time == true)
	{
		for (int i=0; i<VALS_PER_BYTE; i++)
			table[i] = CountOnes(i);
		first_time = false;
	}
	return table[x];
}
```

את הטבלה נחשב באופן חד-פעמי (בעזרת הפונקציה הראשונה), והטרייד אוף יהיה בצריכת הזיכרון.

תשובה לשאלה 82

קריאה ל-Count(N) תחזיר את מספר האפשרויות השונות לטיפוס על בניין בן N קומות:

```
int Count(int num)
{
	if (num<3)
		return num;
	return Count(num-1)+Count(num-2);
}
```

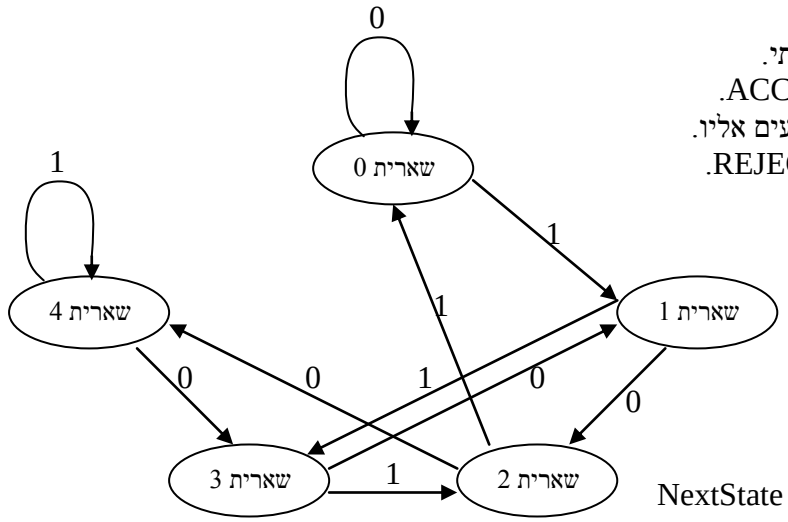
סיבוכיות הזמן היא אקספוננציאלית – O(2^N), כי זה מספר הפעמים שהפונקציה נקראת.
סיבוכיות הזיכרון היא אקספוננציאלית, כי כל קריאה לפונקציה דורשת מקום נוסף במחסנית (שגם היא חלק מהזיכרון).
אפשר לפתור את הבעיה בצורה איטרטיבית במקום בצורה רקורסיבית:

```
int Count(int num)
{
	int prev = 1;
	int curr = 1;
	for (int i=1; i<num; i++)
	{
		int next = prev + curr;
		prev = curr;
		curr = next;
	}
	return curr;
}
```

בפתרון הזה, סיבוכיות הזמן היא לינארית – O(N), וסיבוכיות הזיכרון היא קבועה – O(1).

תשובה לשאלה 83

סה"כ, ישנם חמישה מצבים.
מצב "שארית 0" הוא המצב ההתחלתי.
מצב "שארית 0" הוא גם מצב ACCEPT.
חמישית מהחיצים (2 מתוך 10) מגיעים אליו.
כל שאר המצבים שקולים למצב REJECT.



עבור כל קלט (0 או 1) מתבצע:
NextState = (2 * CurrState + Input) % 5

תשובה לשאלה 84

על מנת לדעת אם קיימת "לולאה" ברשימה מקושרת חד-כיוונית, מספיק למצוא צומת A, שאחריו ברשימה קיים צומת B עם קשת ל-A. מכיוון שמותר להשתמש רק בזיכרון בגודל קבוע, נניח שקיימים שני צמתים כאלה בקטע שבין הצומת הראשון ברשימה לצומת K כלשהו (בהמשך הרשימה), ונחפש אותם באופן הבא:

1. לכל צומת בקטע הצמתים שהגדרנו, נבדוק אם הקשת מצומת K מובילה אליו.
2. אם נמצא צומת כזה, נדע בוודאות שקיימת "לולאה" ברשימה.
3. אם לא, נבדוק אם הקשת מצומת K מובילה לצומת כלשהו.
4. אם לא, נדע בוודאות שלא קיימת "לולאה" ברשימה (כי הגענו לסופה).
5. אם כן, נרחיב את הקטע לצומת הבא אחרי צומת K ונחזור על התהליך שוב.

```
bool SearchLoop(List* list)
{
	for (Node* tail = list->head; tail != NULL; tail = tail->next)
	{
		if (tail->next == tail)
			return true;
		for (Node* curr = list->head; curr != tail; curr = curr->next)
		{
			if (tail->next == curr)
				return true;
		}
	}
	return false;
}
```

סיבוכיות הזמן של האלגוריתם היא O(N^2), וסיבוכיות הזיכרון שלו היא O(1).

שיטה נוספת (באדיבות רותם ויוליה):

- ניעזר בשני מצביעים: ptr1 ו-ptr2. בכל שלב נקדם את ptr1 בצומת אחד ואת ptr2 בשני צמתים.
- אם בשלב כלשהו שניהם יצביעו לאותו צומת, אז נדע שיש "לולאה" ברשימה.

בשיטה הזאת, סיבוכיות הזמן היא O(N) וסיבוכיות הזיכרון היא O(1).

תשובה לשאלה 85 (באדיבות רותם)

אין טריק שמבטיח ניצחון, כי ייתכן שכל המספרים במערך שווים, ואז יהיה תיקו. יש טריק שמבטיח ניצחון או תיקו: השחקן שמתחיל, יכול להביא למצב שבו הוא ייקח את כל האיברים שנמצאים במקומות הזוגיים והיריב שלו ייקח את כל האיברים שנמצאים במקומות האי-זוגיים, או להיפך. לכן, כל מה שהוא צריך לעשות, זה לסכום כל קבוצה בנפרד ולבחור את זאת עם הסכום הגדול ביותר.

לדוגמא, נניח שיש במערך N איברים (כאמור, N מספר זוגי), ושסכום האיברים שנמצאים במקומות הזוגיים גדול יותר מסכום האיברים שנמצאים במקומות האי-זוגיים. על מנת לקחת את כל האיברים שנמצאים במקומות הזוגיים:

- השחקן שמתחיל ראשון לוקח את איבר מספר N, שנמצא במקום זוגי כמובן.
- לאחר מכן, היריב לוקח את איבר מספר 1 או את איבר מספר N-1, שנמצאים שניהם במקומות אי-זוגיים:
  - אם היריב לוקח את איבר מספר 1, השחקן לוקח את איבר מספר 2, שנמצא במקום זוגי.
  - אם היריב לוקח את איבר מספר N-1, השחקן לוקח את איבר מספר N-2, שנמצא גם הוא במקום זוגי.
- לאחר כל איבר שהשחקן לוקח ממקום זוגי, היריב נאלץ לקחת איבר ממקום אי-זוגי.
- לאחר כל איבר שהיריב לוקח ממקום אי-זוגי, השחקן יכול לקחת איבר ממקום זוגי.

תשובה לשאלה 86

על מנת לבדוק אם המספר A הוא חזקה שלמה של שתיים, נתייחס לשני מקרים:

א. A = 0. אפשר להחליט שכן (A שווה לשתיים בחזקת מינוס אינסוף) או שלא.

ב. A > 0. נסתכל על כל הביטים שמייצגים את A, עד ל-MSB (הביט הכי שמאלי שערכו 1):

- A הוא חזקה שלמה של שתיים, אם ורק אם רצף הביטים שמייצג אותו הוא מהצורה 10...0.
- אם נפחית 1 ממספר בעל רצף כזה של ביטים, כולם יתהפכו ונקבל מספר בעל רצף ביטים הפוך: 01...1.
- אם נפחית 1 ממספר בעל רצף אחר של ביטים, לא כולם יתהפכו ולכן לא נקבל מספר בעל רצף ביטים הפוך.
- כלומר, פעולת and בין המספר A למספר A-1 תיתן אפס, אם ורק אם המספר A הוא חזקה שלמה של שתיים.

תשובה לשאלה 87

פונקצית מחלקה מקבלת את הכתובת של האובייקט שקרא לה, אבל בצורה בלתי מפורשת (כלומר, מוסתר מהמתכנת).
את קוד האסמבלי אפשר לבנות, כך שהכתובת של האובייקט תעבור לפונקציה דרך המחסנית או באחד הרגיסטרים.
הקומפיילר של Visual Studio 6.0 בונה את קוד האסמבלי, כך שהיא מקבלת את כתובת האובייקט ברגיסטר ECX.
לפירוט נוסף – חלק 3 (חומר כללי: פונקציות מחלקה).

תשובה לשאלה 88

כל 8 פיקסלים (ביטים) אפשר לייצג באמצעות בית אחד. בכל שורה בתמונה ישנם N פיקסלים, ולכן N/8 בתים.
אפשרות ראשונה:

א. נחשב טבלה שתתאים לכל בית את הבית ההפוך לו:

```
Byte table[1<<8] = {0};
for (int i=0; i<(1<<8); i++)
	for (int j=0; j<8; j++)
		table[i] |= ((i>>j)&1)<<(8-j-1);
```

ב. נעבור על התמונה (מערך דו-מימדי מסוג Byte) ונבצע:

```
for (int m=0; m<M; m++)
	for (int n=0; n<N/8; n++)
		outputImage[m][N/8-n-1] = table[inputImage[m][n]];
```

אפשרות שניה:

א. נגדיר Bit-Field בגודל של בית, שמכיל 8 ביטים:

```
typedef struct
{
	Byte bit1:1;
	Byte bit2:1;
	…
	Byte bit8:1;
}	Bits;
```

ב. נעבור על התמונה (מערך דו-מימדי מסוג Bits) ונבצע:

```
for (int m=0; m<M; m++)
{
	for (int n=0; n<N/8; n++)
	{
		outputImage[m][N/8-n-1].bit8 = inputImage[m][n].bit1;
		outputImage[m][N/8-n-1].bit7 = inputImage[m][n].bit2;
		…
		outputImage[m][N/8-n-1].bit1 = inputImage[m][n].bit8;
	}
}
```

יתרונות וחסרונות:
באפשרות הראשונה נצטרך לחשב טבלה לפני היפוך התמונה, אבל במהלך היפוך התמונה נבצע פחות פעולות קריאה וכתיבה מאשר באפשרות השניה. מצד שני, בהנחה שהטבלה היא הרבה יותר קטנה מהתמונה, סיבוכיות הזמן והזיכרון שכרוכה בה היא שולית. בכל אחת משתי האפשרויות (אם ההנחה הנ"ל נכונה), סיבוכיות הזמן והזיכרון היא O(M*N).

תשובה לשאלה 89 (באדיבות יוליה)

תחילה נמיין את הנקודות הנתונות לקסיקוגרפית לפי קואורדינטות ה-X שלהם. תוך כדי המיון נזכור איזה נקודות מתחילות קטע ואיזה נקודות מסיימות קטע. למשל, ניתן להצמיד מערך של ביטים, כאשר 1 מסמן נקודת התחלה ו-0 מסמן נקודת סיום. במידה ויש כמה נקודות בעלות קואורדינטות X זהות, נבצע מיון מישני באופן הבא – נמקם תחילה את כל הנקודות שמתחילות קטע, ואחריהן את כל הנקודות שמסיימות קטע.

כעת נאתחל Count = MaxCount = 0, נעבור על רשימת הנקודות הממוינת ונבצע:

- אם הנקודה הנוכחית היא נקודה שמתחילה קטע, נגדיל את Count ב-1.
- אם יש צורך, נעדכן את MaxCount.
- אם הנקודה הנוכחית היא נקודה שמסיימת קטע, נקטין את Count ב-1.

מספר נקודות החיתוך המקסימלי הוא הערך של MaxCount בסיום.

הסיבה לביצוע המיון המישני:

- לשם הפשטות, נניח שקיימים רק שני קטעים.
- אם הם סגורים (כלומר, כוללים נקודות קצה), ונקודת הסיום של קטע אחד הינה מעל או מתחת לנקודת ההתחלה של הקטע השני, אז מספר נקודות החיתוך המקסימלי הוא 2. במצב כזה, נרצה קודם להגדיל את Count ב-1 ואחר כך להקטין את Count ב-1 ולא להיפך, כדי שנקבל MaxCount = 2.
- אם הם פתוחים (כלומר, לא כוללים נקודות קצה), ונקודת הסיום של קטע אחד הינה מעל או מתחת לנקודת ההתחלה של הקטע השני, אז מספר נקודות החיתוך המקסימלי הוא 0. במצב כזה, נרצה קודם להקטין את Count ב-1 ואחר כך להגדיל את Count ב-1 ולא להיפך, כדי שנקבל MaxCount = 0.

הערה: בקואורדינטות ה-Y של הנקודות הנתונות אין שימוש כלל.

תשובה לשאלה 90

Constructor של מחלקה כלשהי לא יכול להיות וירטואלי, כי:

קריאה לפונקציה וירטואלית של אובייקט כלשהו מתבצעת בעזרת ה-V-Table של המחלקה שאליה האובייקט שייך.

כל אובייקט מחזיק מצביע ל-V-Table המתאים, אבל המצביע הזה מאותחל רק בזמן ריצה, כאשר האובייקט נוצר.

כלומר, המצביע הזה למעשה מאותחל רק כאשר מתבצעת קריאה ל-Constructor של האובייקט, ולכן ה-Constructor לא יכול להיות וירטואלי. חוץ מזה, אין שום הגיון ב-Constructor וירטואלי. הרעיון מאחורי פונקציות וירטואליות, הוא שאפשר לקרוא להן בלי לדעת מהו הסוג המדויק של האובייקט שבאמצעותו מבצעים את הקריאה. כאשר יוצרים אובייקט (כלומר, קוראים בצורה בלתי מפורשת ל-Constructor), יודעים בדיוק איזה סוג רוצים ליצור, ולכן אין שום צורך במנגנון הזה.

Destructor של מחלקת בסיס (מחלקה שיורשים ממנה) צריך להיות וירטואלי, כי:

כאשר יוצרים ע"י הקצאה סטאטית אובייקט שנגזר ממחלקת הבסיס, אז בסיום הפונקציה (אם האובייקט הוא לוקאלי) או התוכנית (אם האובייקט הוא גלובאלי), ה-Destructor של האובייקט נקרא אוטומטית, ובסיום העבודה שלו הוא גם קורא אוטומטית ל-Destructor של מחלקת הבסיס. במצב כזה, אין משמעות לעובדה שה-Destructor של מחלקת הבסיס הוא וירטואלי. לעומת זאת, כאשר יוצרים ע"י הקצאה דינאמית (new) אובייקט שנגזר ממחלקת הבסיס, אז יש לשחרר אותו בצורה מפורשת (delete). אופרטור ה-delete מקבל את <u>המצביע</u> לאובייקט, שיכול להיות מהסוג של מחלקת הבסיס של האובייקט. במצב כזה, אם ה-Destructor הוא וירטואלי, אופרטור ה-delete קורא ל-Destructor של המחלקה האמיתית של האובייקט, שבסיום העבודה שלו קורא אוטומטית ל-Destructor של מחלקת הבסיס. אחרת, אופרטור ה-delete קורא ל-Destructor של מחלקת הבסיס של האובייקט, וה-Destructor של המחלקה האמיתית של האובייקט לא מתבצע כלל.

לדוגמא, נניח שמחלקת הבסיס היא A, וקיימת מחלקה B שנגזרת ממנה:

עבור A* ptr = new B מתבצע:

- אופרטור ה-new מקבל את הסוג B (בצורה מפורשת), וקורא ל-Constructor של מחלקה B.

עבור delete ptr מתבצע:

- אם ה-Destructor של מחלקה A הוא וירטואלי, אופרטור ה-delete קורא ל-Destructor של מחלקה B.
  כשה-Destructor של מחלקה B מסיים, הוא קורא ל-Destructor של מחלקה A.
- אם ה-Destructor של מחלקה A הוא לא וירטואלי, אופרטור ה-delete קורא ל-Destructor של מחלקה A.
  ה-Destructor של מחלקה B לא נקרא כלל.

לפירוט נוסף – חלק 3 (חומר כללי: פונקציות מחלקה).

תשובה לשאלה 91

על מנת למנוע פניה למערכת ההפעלה בכל פעם שמתבצעת הקצאה דינאמית של אובייקט מסוג A, נגדיר במחלקה A את האופרטור new, אשר יחזיר מצביע לאובייקט מסוג A. מכיוון שאנחנו רוצים למנוע פניה למערכת ההפעלה, האופרטור שנממש יצטרך להחזיר מצביע לאובייקט מסוג A (או מסוג שנגזר מ-A) שכבר קיים במערכת. כלומר, נצטרך להחזיר מצביע לאובייקט שהוקצה בצורה סטאטית. זה לא יכול להיות אובייקט לוקאלי (במחסנית של האופרטור שמימשנו), מכיוון שאסור שפונקציה תחזיר מצביע למשתנה לוקאלי שלה, ולכן זה חייב להיות אובייקט גלובאלי או אובייקט סטאטי במחלקה כלשהי שאינה המחלקה A (כי לא ניתן להגדיר בתוך מחלקה אובייקט מסוג המחלקה עצמה). אפשרות נוספת היא פשוט להחזיר מצביע NULL.

חתימת האופרטור בתוך המחלקה A: `void* operator new(unsigned int stAllocateBlock, char chInit );`

דוגמא לקריאה לאופרטור: `A* a=new(6) A;`

תשובה לשאלה 92

שיטה ליישם ב-C++ את מנגנון הפונקציות הווירטואליות ללא שימוש במילה השמורה virtual:

מחלקה שמספקת ממשק של פונקציות וירטואליות, היא מחלקת בסיס למחלקות אחרות (שיורשות ממנה).

בכל פעם שנגדיר מחלקת בסיס, נוסיף לה משתנה type, שלפיו נדע את הסוג המדויק של כל אובייקט שנוצר.

הערך של type צריך לאפשר לנו לזהות את סוג האובייקט. לדוגמא, הוא יכול להיות מחרוזת עם שם המחלקה שלו.

בכל Constructor של מחלקת הבסיס ושל המחלקות שיורשות ממנה נאתחל את type.

מכיוון שאף Constructor לא יכול להיות וירטואלי, מובטח לנו שכאשר האובייקט נוצר, type יקבל את הערך הנכון.

בפועל, כל Constructor קורא קודם כל ל-Constructor שלפניו בשרשרת ההורשה, ורק אז מבצע את הקוד שלו עצמו. כלומר, לפעמים המשתנה type יקבל ערך יותר מפעם אחת, אבל הערך האחרון שהוא יקבל יהיה הערך הנכון.

למשל, נניח שמחלקה A היא מחלקת הבסיס ומחלקות B ו-C יורשות ממנה.

מכיוון שלא ניתן להשתמש במילה השמורה virtual, כל קריאה לפונקציה של A בעזרת מצביע מסוג A, תביא אותנו לפונקציה של A גם אם העצם המוצבע הוא מסוג B או C.

לכן, בכל פונקציה של A שנרצה שתהיה וירטואלית, נבדוק את הערך של type ונזהה את סוג המחלקה שהפונקציה שלה צריכה להתבצע. אם זיהינו שסוג האובייקט הוא A, נבצע את הקוד שהגדרנו עבור המחלקה A (אם אין הגדרה כזאת, לא נבצע כלום). אם זיהינו שסוג האובייקט הוא B או C, נמיר את ה-this שברשותנו למצביע מתאים, ונקרא בעזרתו לפונקציה של המחלקה המתאימה.

מכיוון שאנחנו יודעים בוודאות מהו סוג האובייקט שלנו, ההמרה של ה-this למצביע מהסוג הזה והקריאה לפונקציה בעזרתו יהיו "חוקיות" (לא נקבל שגיאה בזמן ריצה).

דוגמא למימוש של פונקציה "וירטואלית" במחלקה A:

```
int A::func()
{
	if (type == 'A')
		return 0;
	else if (type == 'B')
		return ((B*)this)->func();
	else //if (type == 'C')
		return ((C*)this)->func();
}
```

הערות:

א. השימוש במשתנה type ובהמרת המצביעים (כמו בדוגמא הנ"ל) דומה לשימוש באופרטור dynamic_cast, שבזמן ריצה בודק את הסוג האמיתי של העצם המוצבע לפני שהוא ממיר את המצביע לסוג הנדרש. אם הבדיקה מראה שהם זהים, אז האופרטור מחזיר מצביע מתאים. אחרת, הוא מחזיר NULL. על מנת להשתמש באופרטור הזה, צריך לוודא שקיימת בקומפיילר אופציית RTTI (Run-Time Type Information), שמאפשרת לקבל אינפורמציה לגבי סוג האובייקט בזמן ריצה.

ב. באופן כללי, גם השימוש ב-RTTI וגם השיטה הנ"ל ליישום של מנגנון הפונקציות הווירטואליות, נוגדים את הקונספט של OO. הרעיון בקונספט הזה הוא הכללה של אובייקטים דומים, ושימוש בהם בלי לדעת מהו הסוג המדויק של כל אחד (Polymorphism). כאן עשינו בדיוק את ההיפך – בדקנו את סוג האובייקט וביצענו פעולה מסוימת בהתאם.

ג. בנוסף, בכל פעם שנרצה לגזור (לרשת) ממחלקת הבסיס מחלקה חדשה, נצטרך לשנות את קוד הפונקציות ה"ווירטואליות" של מחלקת הבסיס. כלומר, נצטרך לקמפל וללנקג' גם את קוד המקור של מחלקת הבסיס (בנוסף לקוד המקור של המחלקה החדשה).

תשובה לשאלה 93

נניח לשם הפשטות שיש במערך מספר זוגי של איברים:

- נחלק את המערך לזוגות של מספרים, ולכל זוג מספרים נמצא את המינימום ואת המקסימום.
- מכיוון שלכל זוג מספרים מספיק לבצע השוואה אחת (if x<y), נבצע סך הכל ½N השוואות.

לאחר שקיבלנו ½N ערכי מינימום ו-N ½ ערכי מקסימום:

- נעבור על ערכי המינימום ונמצא את הערך המינימלי מביניהם – סך הכל ½N השוואות.
- נעבור על ערכי המקסימום ונמצא את הערך המקסימלי מביניהם – סך הכל ½N השוואות.

סך הכל, מספר ההשוואות שביצענו הוא ½N + ½N + ½N = 1½N.

תשובה לשאלה 94

ניעזר בטבלה בגודל M על N, על מנת לפתור את הבעיה בעזרת תכנון דינאמי:

א. כל משבצת בטבלה תייצג את הריבוע המקסימלי של פיקסלים מאותו צבע, שהפינה הימנית התחתונה שלו היא הפיקסל המקביל בתמונה. כלומר, בכל משבצת נרשום את אורך הצלע של הריבוע שאותו היא מייצגת.

ב. בכל המשבצות בשורה הראשונה ובטור הראשון בטבלה נרשום 1, מכיוון שזהו אורך הצלע של הריבוע שכל אחת מהן מייצגת.

ג. נעבור על הטבלה בצורה סדרתית (משמאל לימין בכל שורה, לפי סדר השורות), ולכל משבצת נחשב את הערך המתאים לה באופן הבא:

- נבדוק את הצבע של משבצת [i][j] ושל 3 המשבצות [i][j-1] , [i-1][j] , [i-1][j-1] .
- אם הצבעים של חלק מ-4 המשבצות שונים זה מזה, אז משבצת [i][j] מייצגת ריבוע שאורך הצלע שלו הוא 1.
- אם הצבעים של כל 4 המשבצות זהים זה לזה, אז משבצת [i][j] מייצגת ריבוע שאורך הצלע שלו הוא 1 + הערך המינימלי מבין הערכים שרשומים ב-3 המשבצות [i][j-1] , [i-1][j] , [i-1][j-1] .

ד. לסיום, נעבור על הטבלה ונמצא את הערך המקסימלי שרשום בה.

נניח, למשל, שהתמונה נראית כך:

R	R	G	G	B
G	G	G	G	G
G	G	G	G	G
B	B	B	G	G
B	B	B	G	G

כאשר נגיע למשבצת הרביעית בשורה השלישית, הטבלה תיראה כך:

1	1	1	1	1
1	1	**1**	**2**	1
1	2	**2**	x	
1				
1				

נחשב את הערך של המשבצת הזאת כך: x = 1+Min(1,2,2) = 2.

תשובה לשאלה 95

בפונקציה MyMalloc נקצה 4 בתים מעבר לכמות הבתים הנדרשת, ונרשום בהם את מספר הבתים שהוקצו:

```
void* MyMalloc(int size)
{
	int* mem = malloc(sizeof(int)+size);
	mem[0] = sizeof(int)+size;
	return (void*)(mem+1);
}
```

בפונקציה MyFree נקרא מ-4 הבתים הראשונים את מספר הבתים שהוקצו, ונשחרר את כמות הבתים הזאת:

```
void MyFree(void* ptr)
{
	int* mem = (int*)ptr-1;
	int size = mem[0];
	free((void*)mem,size);
}
```

תשובה לשאלה 96

ניהול החוצצים יתבצע באופן הבא:

- נייצג כל חוצץ באמצעות מבנה נתונים "באפר", שכולל כתובת זיכרון ודגל "פנוי" / "תפוס".
- נקצה N חוצצים, ומחסנית שיכולה להכיל N "באפרים". מכיוון שהתוכנית היא תוכנית Real-Time, נבצע את כל ההקצאות בזמן קומפילציה במקום בזמן ריצה (הקצאה סטאטית במקום הקצאה דינאמית), ע"מ להבטיח שני דברים:
  א. כל אלמנט יקבל מקום בזיכרון (או שהתוכנית לא תעבור קומפילציה, ואז נדע שיש בעיה).
  ב. כל אלמנט יישב באותו מקום בזיכרון, וזמן הגישה אליו לא ישתנה בכל פעם שהתוכנית תרוץ.
- הפונקציה void InitBuffers(void* addresses[N]):
  נאתחל N "באפרים" בכתובות של החוצצים ובדגלי "פנוי", ונכניס אותם למחסנית – O(N).
- הפונקציה Buffer* GetBuffer():
  נוודא שהמחסנית לא ריקה, נוציא ממנה "באפר", נשנה את הדגל שלו ל"תפוס" ונחזיר אותו כפלט – O(1).
- הפונקציה void FreeBuffer(Buffer* buffer):
  נוודא שהדגל של ה"באפר" שקיבלנו כקלט הוא "תפוס", נשנה אותו ל"פנוי" ונכניס אותו למחסנית – O(1).

שינוי השדות של מבנה הנתונים "באפר" יתאפשר רק בתוך הפונקציות הנ"ל.

תשובה לשאלה 97

הכפלת רגיסטר R1 ברגיסטר R2, והכנסת התוצאה לרגיסטר R3:

```
CLR R3                  //R3 = 0
CLR R7                  //R7 = 0
CLR R8                  //R8 = 0
Loop_R1:
DEC R1                  //R1 = R1 - 1
INC R7                  //R7 = R7 + 1
Loop_R2:
DEC R2                  //R2 = R2 - 1
INC R8                  //R8 = R8 + 1
INC R3                  //R3 = R3 + 1
JUMP R2 Loop_R2         //if (R2 > 0) goto Loop_R2
Restore_R2:
DEC R8                  //R8 = R8 - 1
INC R2                  //R2 = R2 + 1
JUMP R8 Restore_R2      //if (R8 > 0) goto Restore_R2
JUMP R1 Loop_R1         //if (R1 > 0) goto Loop_R1
Restore_R1:
DEC R7                  //R7 = R7 - 1
INC R1                  //R1 = R1 + 1
JUMP R7 Restore_R1      //if (R7 > 0) goto Restore_R1
```

הערות:

- אם קלט אחד הוא 0, אז הקלט השני יוכפל ב-16.
- מכיוון שהפלט נתון ברגיסטר של 4 ביטים, התוצאה עדיין תהיה 0.

פעולות שניתן לממש באמצעות פעולות אחרות:

- CLR ניתן לממש באמצעות DEC ו-JUMP.
- INC ניתן לממש באמצעות DEC ו-JUMP (מכיוון שביצוע INC שקול לביצוע DEC חמש עשרה פעמים).

הקוד עובד נכון גם עבור מספרים שליליים:

- ישנן רק 8 מכפלות אפשריות שבהן אין גלישה.
- מספיק לבדוק אותו עבור כל אחת מהמכפלות האלה: 1*1-, 1*2-, 1*3-, 1*4-, 1*5-, 1*6-, 1*7-, 1*8-.

תשובה לשאלה 100

פונקצית מחלקה סטאטית לא יכולה להיות וירטואלית. הסיבה לכך, היא שהקריאה לפונקציה וירטואלית מתבצעת על סמך סוג האובייקט שבאמצעותו הפונקציה נקראת, ושאת המצביע אליו היא מקבלת (this). פונקציה סטאטית לא מקבלת מצביע לאובייקט שבאמצעותו היא נקראת, ולכן אין משמעות לסוג שלו. למעשה, אפשר באותה מידה לקרוא לה באמצעות המחלקה שאליה היא שייכת. לדוגמא, הקריאה MyObject->Func() שקולה לקריאה MyClass::Func().
לפירוט נוסף – חלק 3 (חומר כללי: פונקציות מחלקה).

תשובה לשאלה 101 (באדיבות עדי)

המסלול הקצר ביותר מהספינה אל האי נוגע רק בקודקודים של הקרחונים, ולא בצלעות שלהם (דבר שנובע מהעובדה, שהמרחק הקצר ביותר בין כל שתי נקודות הוא הקו הישר שמחבר ביניהן).
נגדיר אוסף של צמתים שמייצגים את הנקודה שבה נמצאת הספינה, הנקודה שבה נמצא האי והנקודות שבהן נמצאים הקרחונים (הקודקודים של המצולעים שמייצגים אותם).
בין כל שני צמתים שמייצגים נקודות שיש ביניהן קו ישר חוקי (כזה שלא עובר דרך קרחון) נגדיר קשת, ונרשום עליה את אורך הקו שהיא מייצגת.
על הגרף שנוצר נבצע דייקסטרה, החל מהצומת שמייצג את הספינה:

- במהלך האלגוריתם, נרשום בכל צומת שאליו נגיע את המרחק הקצר ביותר מהספינה. בנוסף, נרשום בו מצביע לצומת שממנו נצטרך להגיע על מנת להשיג את המרחק הזה.
- בסוף האלגוריתם, נקבל בצומת שמייצג את האי את המרחק הקצר ביותר מהספינה. בנוסף, נוכל לשחזר את המסלול שאותו נצטרך לעשות מהספינה על מנת להשיג את המרחק הזה.

תשובה לשאלה 102

כמות החלב שווה לכמות סירופ השוקולד. לכן, לדרך שבה נערבב את שתי התכולות לא תהיה כל השפעה על התוצאה הסופית: כל עוד כמות החומר במיכל הראשון שווה לכמות החומר במיכל השני, ריכוז החלב במיכל הראשון יהיה שווה לריכוז סירופ השוקולד במיכל השני.

תשובה לשאלה 103

מספיק שאחד האנשים יוכל לחשב את ממוצע המשכורות. נבחר את האיש הראשון לצורך כך:

- האיש הראשון בוחר ערך x כלשהו, מחבר אליו את המשכורת שלו ואומר את התוצאה לאיש השני.
- האיש השני מחבר אל הערך שקיבל מהאיש הראשון את המשכורת שלו ואומר את התוצאה לאיש השלישי.
- האיש השלישי מחבר אל הערך שקיבל מהאיש השני את המשכורת שלו ואומר את התוצאה לאיש הראשון.
- האיש הראשון מחסר מהערך שקיבל מהאיש השלישי את הערך x, מחלק את התוצאה ב-3 ומקבל את הממוצע.

תשובה לשאלה 104

נניח שמספר הקלט הוא x, ומספר הביטים ב-x שערכם 1 הוא N. על מנת למצוא את N תוך שימוש ב-N איטרציות:

א. בכל איטרציה נאפס ב-x את הביט הימני ביותר שערכו 1, באופן הבא:
   - 10...0 זה הייצוג הבינארי של x, מה-LSB ועד לביט הראשון שערכו 1.
   - 01...1 זה הייצוג הבינארי של x-1, מה-LSB ועד לביט הראשון שערכו 0.
   - נבצע פעולת and בין הערך של x לערך של x-1, ונכניס את התוצאה לתוך x.
   - כתוצאה מכך, הביט הראשון ב-x שערכו 1 יהפוך ל-0, ושאר הביטים ב-x לא ישתנו.

ב. כאשר הערך של x יגיע לאפס, נדע שמספר האיטרציות שביצענו הוא הערך המבוקש של N.

תשובה לשאלה 105

על מנת למצוא את ערך המספר שמופיע במערך פעמיים:

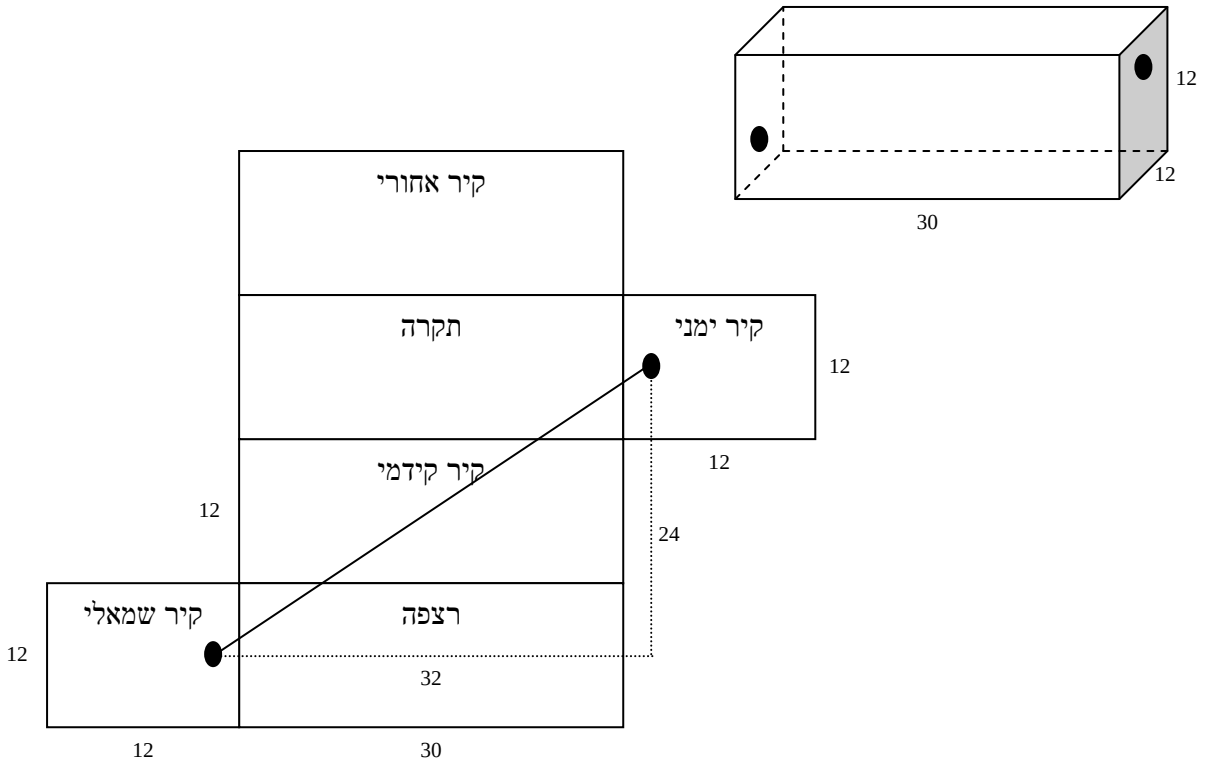
1. נעבור על המערך פעם אחת ונחשב את הסכום של N המספרים שבו.
2. הסכום של N-1 המספרים שבתחום [1, N-1] הוא 1+2+…+N-1 = N*(N-1)/2.
3. נפחית מהסכום של N המספרים שבמערך את הסכום של N-1 המספרים שבתחום [1, N-1].

תשובה לשאלה 106

הכפלה של x ב-7, ללא שימוש בפעולת כפל או בפעולת חיבור: x*7 = x*(8-1) = x*8-x = (x<<3)-x.

תשובה לשאלה 107

נסתכל על החדר כעל קופסת קרטון תלת-מימדית, שאותה נפרוס למשטח דו-מימדי באופן הבא:



במשטח שקיבלנו, נחבר את שני השקעים בעזרת קו ישר.

הכבל יעבור דרך 5 (מתוך 6) פאות של החדר. אפשר לחשב את אורכו לפי משפט פיטגורס: $\sqrt{(32^2+24^2)} = 40$.

תשובה לשאלה 108

הפתרון הטכני:

- הגבר הראשון צריך לשים את הקונדום הראשון ועליו את הקונדום השני.
- הגבר השני צריך לשים את הקונדום השני.
- הגבר השלישי צריך לשים את הקונדום הראשון הפוך ועליו את הקונדום השני.

צורה אחרת להסתכל על זה – ישנם שלושה גברים, אשה אחת ושני קונדומים. לכל קונדום יש שני צדדים.

בסך הכל ישנם ארבעה אנשים וארבעה צדדים, ולכן צריך התאמה חד-ערכית בין האנשים והצדדים. בפתרון הנ"ל:

- הגבר הראשון מקבל את הצד הפנימי של הקונדום הראשון.
- הגבר השני מקבל את הצד הפנימי של הקונדום השני.
- הגבר השלישי מקבל את הצד החיצוני של הקונדום הראשון.
- האשה מקבלת את הצד החיצוני של הקונדום השני.

תשובה לשאלה 109

נסמן ב-x את מספר הקלט ונעבור על הביטים שלו (מה-LSB ל-MSB):

- אם ערך הביט הוא 1, נשנה אותו ל-0 ונמשיך לביט הבא.
- אם ערך הביט הוא 0, נשנה אותו ל-1 ונסיים את הלולאה.

```
for (i=1; i!=0; i<<=1)
{
	if (x & i)		//The bit value is 1
		x &= ~i;
	else			//The bit value is 0
	{
		x |= i;
		break;
	}
}
```

תשובה לשאלה 110

נשתמש במערך עזר בגודל N (נקרא לסדרת הקלט List ולמערך העזר Array).
בכל איבר במערך העזר נרשום את הסכום של תת הסדרה הרצופה המקסימלית שמסתיימת בו.
את המערך נמלא בצורה אינדוקטיבית (כלומר, נחשב כל ערך על סמך הערך הקודם), באופן הבא:

א. עבור האיבר הראשון: Array[1] = List[1]
כי הסכום של תת הסדרה הרצופה היחידה שמסתיימת באיבר הראשון שווה לערך של האיבר עצמו.

ב. עבור כל איבר אחר: Array[i] = Max(Array[i-1]+List[i], List[i])
כי הסכום של תת הסדרה הרצופה המקסימלית שמסתיימת בכל איבר אחר שווה לערך הגדול מבין:
- הסכום של תת הסדרה שמסתיימת באיבר הקודם + הערך של האיבר עצמו.
- הערך של האיבר עצמו (אם הסכום של תת הסדרה שמסתיימת באיבר הקודם הוא קטן מאפס).

לסיום, נעבור על מערך העזר ונמצא את הסכום של תת הסדרה הרצופה המקסימלית שקיימת בסדרת הקלט.
סיבוכיות זמן: O(N). סיבוכיות זיכרון: O(N).
הערה:
בשיטה הזאת ניתן למצוא את <u>הסכום</u> של תת הסדרה. על מנת למצוא את תת הסדרה עצמה (כלומר, את אינדקס ההתחלה ואת אינדקס הסיום שלה), נצטרך לשמור בכל תא במערך העזר גם את אינדקס ההתחלה שלה. אינדקס הסיום שלה יהיה, כמובן, האינדקס של התא עצמו.

תשובה לשאלה 111

נניח שלפני הסיבוב, המערך ממוין בסדר עולה. אחרי הסיבוב, המערך מורכב משני חלקים. כל חלק ממוין בסדר עולה, אבל בנקודת החיבור שלהם הסדר לא מתקיים. על מנת למצוא את נקודת החיבור ביניהם, נבצע את האלגוריתם הבא:

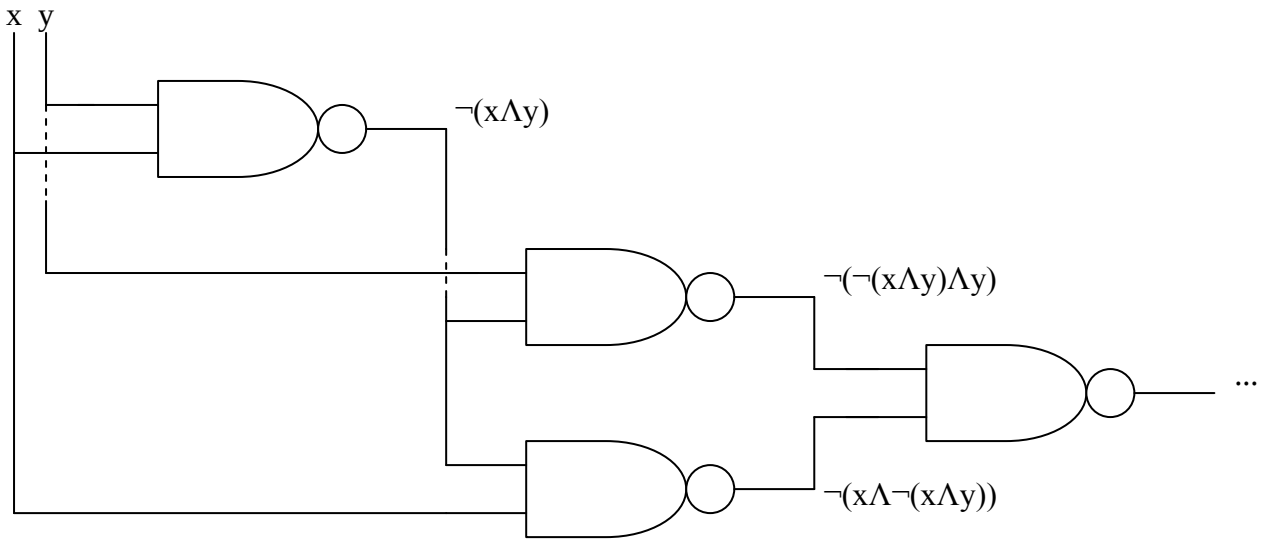
א. נחלק את המערך לשני חצאים.
- בחצי אחד האיבר הראשון קטן מהאיבר האחרון, והחצי הזה ממוין בסדר עולה.
- בחצי השני האיבר הראשון גדול מהאיבר האחרון, והחצי הזה לא ממוין בסדר עולה.

ב. נבחר את החצי הלא ממוין, ונחזור על התהליך עד שנגיע ל"חצי" שמכיל שני איברים בלבד (שם נקודת החיבור בין החלקים). מכיוון שכל חלק ממוין בסדר עולה, בהינתן איבר כלשהו, נוכל לבצע חיפוש בינארי על כל חלק בנפרד.

סיבוכיות זמן:
האלגוריתם למציאת נקודת החיבור בין החלקים פועל בדומה לחיפוש בינארי, וסיבוכיות הזמן שלו היא O(log(N)).
על מנת למצוא איבר כלשהו, נבצע חיפוש בינארי על כל חלק בנפרד, ולכן סיבוכיות הזמן הכוללת תישאר O(log(N)).

תשובה לשאלה 112

נבנה שער XOR בעזרת ארבעה שערי NAND באופן הבא:



הוכחה:
$\neg(\neg(x\Lambda\neg(x\Lambda y))\Lambda\neg(\neg(x\Lambda y)\Lambda y)) = (x\Lambda\neg(x\Lambda y))V(\neg(x\Lambda y)\Lambda y) = (x\Lambda\neg x)V(x\Lambda\neg y)V(\neg x\Lambda y)V(\neg y\Lambda y) = \underline{(x\Lambda\neg y)V(\neg x\Lambda y)}$

תשובה לשאלה 113

כאשר אצן A ישלים 100 מטר, אצן B ישלים 95 מטר (כמו במירוץ הקודם).
מכיוון שאצן A מתחיל 5 מטר מאחורי אצן B, שניהם יגיעו בדיוק לאותה נקודה, ולכל אחד מהם יישארו 5 מטר לסיום.
את 5 המטרים האחרונים אצן A יסיים לפני אצן B (כי הוא יותר מהיר ממנו).

תשובה לשאלה 114

במצב הנתון:

- בבקבוק של תרופה X נשארו 29 כדורים.
- בבקבוק של תרופה Y נשארו 28 כדורים.
- ביד של החולה יש כדור אחד מסוג X ושני כדורים מסוג Y.

על מנת לקחת את התרופות לפי ההוראות של הרופא:

- החולה צריך להוציא כדור אחד מהבקבוק של תרופה X:
  - בבקבוק של תרופה X יישארו 28 כדורים.
  - בבקבוק של תרופה Y יישארו 28 כדורים.
  - ביד של החולה יהיו שני כדורים מסוג X ושני כדורים מסוג Y.
- לאחר מכן, הוא צריך לשים את ארבעת הכדורים בטור ולחצות אותם לשתיים.
- ביום הראשון הוא צריך לקחת את ארבעת החצאים שנמצאים בצד הימני של הטור.
- ביום השני הוא צריך לקחת את ארבעת החצאים שנמצאים בצד השמאלי של הטור.
- בכל יום לאחר מכן (במהלך 28 הימים שנותרו) הוא צריך לקחת כדור אחד מכל בקבוק.

תשובה לשאלה 115

נתון מערך קלט של N ערכים שלמים בתחום [0…N-1].
נגדיר מערך בוליאני בגודל N, ונאתחל את כל הערכים בו ל-false.
לכל ערך X במערך הקלט, נבדוק את הערך של תא מספר X במערך הבוליאני.
אם הוא false, נשנה אותו ל-true, ונמשיך. אחרת, נדע שהערך X מופיע יותר מפעם אחת במערך הקלט, ונעצור.
לאחר שהמשכנו N איטרציות בלי שעצרנו, נדע שכל ערך בתחום [0…N-1] מופיע במערך הקלט בדיוק פעם אחת.
סיבוכיות זמן: O(N). סיבוכיות זיכרון: O(N).

```
bool func(int Array[N])
{
	bool Test[N] = {false};
	for (int i=0; i<N; i++)
		if (Test[Array[i]] == true)
			return true;
	return false;
}
```

תשובה לשאלה 116

ניתוח הבעיה:

- שחקן שרוצה להביא את הקופה ל-X מטבעות, צריך שיהיו בקופה X-4 מטבעות.
- במצב כזה היריב שלו ייאלץ לשים בקופה 1, 2 או 3 מטבעות, והוא יוכל להשלים את החסר.
- באותו אופן, שחקן שרוצה שיהיו בקופה X-4 מטבעות, צריך שיהיו בקופה X-8 מטבעות (וכן הלאה).

השיטה הכללית:

- השחקן שמתחיל ראשון צריך לשים בקופה X%4 מטבעות.
- לאחר מכן, מספר המטבעות שנשאר לשים בקופה מתחלק ב-4.
- לכן, הוא צריך "להשלים" כל כמות מטבעות שהיריב שלו שם ל-4.

תיאור האלגוריתם:

- X לא מתחלק בשלמות ב-4, ולכן מתקיים $1 \le X\%4 \le 3$.
- השחקן שמתחיל ראשון צריך לשים בקופה X%4 (1, 2 או 3) מטבעות.
- היריב שלו יכול לשים בקופה Y מטבעות. $1 \le Y \le 3$, ולכן מתקיים $1 \le 4-Y \le 3$.
- לאחר כל פעם שהיריב שלו שם בקופה Y מטבעות, הוא צריך לשים בקופה 4-Y (1, 2 או 3) מטבעות.

תשובה לשאלה 117

מספר הפעמים שהספרה 0 מופיעה ברציפות בסוף הייצוג העשרוני של X, שווה למספר הפעמים ש-X מתחלק ב-10.
שני הגורמים הראשוניים של 10 הם 2 ו-5, ולכן מספר הפעמים ש-X מתחלק ב-10 שווה לערך הקטן מבין שני אלה:

- מספר הפעמים ש-X מתחלק ב-2.
- מספר הפעמים ש-X מתחלק ב-5.

100 עצרת "מכיל" 20 גורמים שמתחלקים ב-5 לפחות פעם אחת.
4 גורמים מתוכם (25, 50, 75, 100) מתחלקים ב-5 אפילו פעמיים.
כלומר, בסך הכל, 100 עצרת מתחלק ב-5 בדיוק עשרים וארבע פעמים.
100 עצרת "מכיל" 50 גורמים שמתחלקים ב-2 לפחות פעם אחת (כלומר, 5 הוא "צוואר הבקבוק" בהקשר הזה).
מסקנה: 100 עצרת מתחלק ב-10 בדיוק 24 פעמים, והספרה 0 מופיעה 24 פעמים ברציפות בסוף הייצוג העשרוני שלו.

תשובה לשאלה 118

כאשר המערך הוקצה בצורה סטאטית, Type Array[10], מספר האיברים בו הוא sizeof(Array)/sizeof(Type).
כאשר המערך הוקצה בצורה דינאמית, Type Array=new Type[10], לא ניתן לדעת מה מספר האיברים שהוא מכיל.

תשובה לשאלה 119

בקטע הקוד T* p=0; delete p; יתבצע ניסיון לשחרר sizeof(T) בתים מכתובת 0 בזיכרון.
אם ממומש אופרטור delete, שמקבל מצביע לעצם מסוג T ובודק את הערך שלו לפני שהוא משחרר את הזיכרון שהעצם תופס, אז קטע הקוד הזה יכול להסתיים בצורה תקינה. אחרת, סביר להניח שהוא יגרום לקריסת התוכנית.
בקטע הקוד T* p=new T[10]; delete p; ה-Destructor של T לא ייקרא עבור כל איבר במערך p.
אם T הוא טיפוס פרימיטיבי כלשהו (למשל, int), אז זה לא ישנה, אבל אם T היא מחלקה שבעצמה מבצעת שחרור של זיכרון (ב-Destructor שלה), אז התוצאה תהיה "דליפת זיכרון". הפתרון הוא לבצע "delete[] p" במקום "delete p".

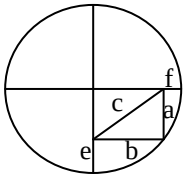
תשובה לשאלה 120

RTTI – Run-Time Type Information:
אופציית קומפיילר, שמאפשרת לבדוק את הסוג המדויק של אובייקטים בזמן ריצה. את הבדיקה ניתן לבצע באמצעות האופרטור dynamic_cast, שבזמן ריצה בודק את הסוג האמיתי של העצם המוצבע לפני שהוא ממיר את המצביע לסוג הנדרש. אם הבדיקה מראה שהם זהים, אז האופרטור מחזיר מצביע מתאים. אחרת, הוא מחזיר NULL.
בדוגמא הבאה, A היא מחלקת בסיס, ו-B ו-C הן מחלקות שנגזרות ממנה:

```
A* arr[3];
arr[0] = new A;
arr[1] = new B;
arr[2] = new C;
for (int i=0; i<3; i++)
{
	B* ptr = dynamic_cast<B*>(arr[i]);
	if (ptr != NULL)
		;	//arr[i] is of type B
	else
		;	//arr[i] is not of type B
}
```

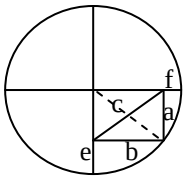
אני לא בטוח, אבל נראה לי שזה עובד ככה:
הקומפיילר מייצר קוד אסמבלי (עבור שימוש באופרטור dynamic_cast), שמבצע בדיקה של סוג האובייקט. הבדיקה עצמה מתבצעת בזמן ריצה, על ידי השוואה בין כתובת ה-V-Table של הסוג הנדרש (ידועה כבר בזמן קומפילציה) לכתובת ה-V-Table של האובייקט עצמו (ידועה רק בזמן ריצה, לאחר שהאובייקט נוצר ומקבל את הערך שלה).
אני חושב שבגלל זה אפשר להשתמש באופרטור dynamic_cast רק עם מחלקות שיש להן פונקציות וירטואליות. שימוש באופרטור הזה עם מחלקה שאין לה פונקציות וירטואליות לא עובר קומפילציה (כי אין לה V-Table).
הערה:
באופן כללי, השימוש ב-RTTI (ובאופרטור dynamic_cast) נוגד את הקונספט של OO. הרעיון בקונספט הזה הוא הכללה של אובייקטים דומים, ושימוש בהם בלי לדעת מהו הסוג המדויק של כל אובייקט ואובייקט (Polymorphism).

תשובה לשאלה 121



חישוב קוטר המעגל כפונקציה של המשתנים a, b, c, e, f:

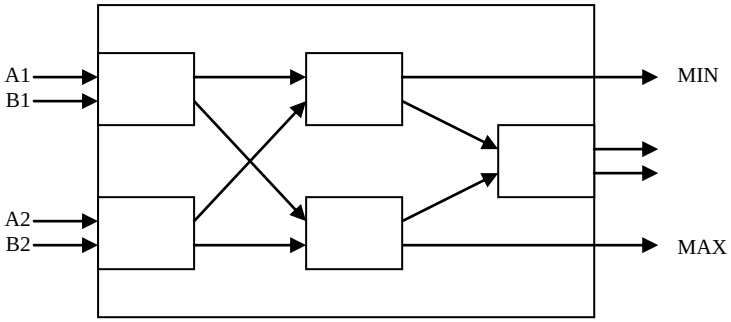
- רדיוס המעגל שווה a+e וגם b+f.
- לכן, קוטר המעגל שווה 2a+2e וגם 2b+2f.



חישוב קוטר המעגל כפונקציה של המשתנים a, b, c בלבד:

- נשרטט את הרדיוס ממרכז המעגל לקודקוד המשולש.
- רדיוס המעגל הזה שווה c (אלכסונים במלבן).
- לכן, קוטר המעגל שווה 2c.

תשובה לשאלה 122



- נמיין את זוג מספר 1.
- נמיין את זוג מספר 2.
- נמיין את המינימליים מכל זוג.
- נמיין את המקסימליים מכל זוג.
- נמיין את המינימלי הגדול והמקסימלי הקטן.

סה"כ, השתמשנו בחמש יחידות.

תשובה לשאלה 124

פתיל A מתכלה תוך 60 דקות, ופתיל B מתכלה תוך 30 דקות:

- אם נדליק את פתיל A משני הצדדים בו-זמנית, שתי הלהבות ייפגשו בנקודה כלשהי תוך 30 דקות, והפתיל יתכלה.
- אם נדליק את פתיל B משני הצדדים בו-זמנית, שתי הלהבות ייפגשו בנקודה כלשהי תוך 15 דקות, והפתיל יתכלה.
- על מנת למדוד 45 דקות בעזרת פתילים A ו-B:
  1. נדליק את פתיל A משני הצדדים בו-זמנית, וכאשר הוא יתכלה, נדע שעברו 30 דקות.
  2. לאחר מכן, נדליק את פתיל B משני הצדדים בו-זמנית, וכאשר הוא יתכלה, נדע שעברו עוד 15 דקות.

פתיל C מתכלה תוך 60 דקות:

- על מנת למדוד 45 דקות בעזרת פתילים A ו-C:
  1. נדליק את פתיל A משני הצדדים ואת פתיל C מצד אחד בו-זמנית.
  2. לאחר 30 דקות פתיל A יתכלה, ולפתיל C יישארו 30 דקות נוספות לבעור.
  3. נדליק את פתיל C מהצד השני שלו, וכאשר הוא יתכלה, נדע שעברו עוד 15 דקות.

תשובה לשאלה 125

פונקציה שמקבלת בית, ומחזירה את הבית ההפוך לו מבחינת סדר הביטים שלו:

- קוראים את הביטים של בית הקלט.
- כותבים אותם בסדר הפוך בבית הפלט.

```
Byte ReverseByte(Byte input)
{
	Byte output = 0;
	for (int i=0; i<8; i++)
	{
		Byte unit = (input>>i)&1;
		output |= unit<<(8-i-1);
	}
	return output;
}
```

פונקציה שעושה את אותה פעולה עבור בית, בזמן קבוע ביחס למספר הביטים שלו:

- מחשבים באופן חד-פעמי טבלה שממפה כל בית לבית ההפוך לו.
- מוצאים בטבלה את הערך המתאים לקלט ומחזירים אותו כפלט.

```
Byte ReverseByteFast(Byte input)
{
	static bool first_time = true;
	static Byte table[1<<8] = {0};
	if (first_time == true)
	{
		for (int i=0; i<(1<<8); i++)
			table[i] = ReverseByte(i);
		first_time = false;
	}
	return table[input];
}
```

פונקציה שעושה את אותה פעולה עבור רגיסטר, בזמן קבוע ביחס למספר הביטים שלו:

- קוראים את הבתים של רגיסטר הקלט.
- כותבים אותם הפוכים ובסדר הפוך ברגיסטר הפלט.

```
Register ReverseRegisterFast(Register input)
{
	Register output = 0;
	for (int i=0; i<sizeof(Register); i++)
	{
		Register unit = ReverseByteFast(input>>(8*i));
		output |= unit<<(8*(sizeof(Register)-i-1));
	}
	return output;
}
```