

חומר טכני לראיון עבודה

1. תיאור כללי:

- חלק "צפוי".
- חלק "בלתי צפוי".
- דוגמא לתיאור של פרויקט.

2. רשימת שאלות:

- אינטל.
- רפא"ל.
- חברה שאני לא זוכר את השם שלה.
- בי.איי.טי.אם.
- איי.בי.אם.
- אלביט.
- מיקרוסופט.
- קיי.אל.איי.
- צורן.
- ספקטרום דיינאמיקס.
- פריסקייל.
- שאלות ששמעתי מאחרים.

3. חומר כללי:

- סמלים בתוכנה.
- פונקציות מחלקה.
- תכנות מונחה עצמים.
- ניהול שכבות הזיכרון.
- ניהול תהליכים וחוטים.

4. תשובות לחלק מהשאלות.

כל מה שכתוב כאן מתבסס על הידע והניסיון שלי (אז קחו את זה בעירבון מוגבל). barakman@yahoo.com.

חלק "צפוי"

1. בחלק הזה אתה מתבקש לתאר פרוייקט כלשהו שעשית. סביר להניח שאם אתה סטודנט ללא נסיון עבודה, אז זה יהיה פרוייקט מהלימודים, ואם יש לך נסיון קודם בעבודה, אז זה יהיה פרוייקט שעבדת עליו.
2. 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.
3. נקודה חשובה נוספת: ייתכן שגם תישאל לגבי דברים מסויימים שציננת (בקורות החיים שלך) שאתה יודע או שיש לך נסיון בהם. לכן, במידה ויש כאלה, כדאי להתכונן ולחזור גם עליהם.

חלק "בלתי צפוי"

4. בחלק הזה שואלים אותך שאלות טכניות או חידות. השאלות הטכניות יכולות להיות דברים ספציפיים שקשורים לתוכנה או חומרה, מה-Low Level (המעבד או מערכת ההפעלה) ועד ה-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 אותן לבעיה ואז לפתור אותן באופן דומה (פתרון אלגוריתמי).
5. 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.
6. מהנסיון שלי, לרוב הבעיות יש פתרון פשוט ובנאלי, שהחסרון שלו הוא שהוא איטי או דורש הרבה זכרון (סיבוכיות זמן או מקום גבוהה). כדאי בדרך כלל להציג את הפתרון הזה בתור התחלה (אם מזהים אחד כזה, כמובן) ולהדגיש שזה הפתרון המידי שעולה לראש. אחרי זה אפשר לציין את החסרונות שלו, ואז לנסות למצוא (ביחד עם המראיין) שיפורים.

דוגמא לתיאור של פרויקט

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

ריכוזי כמה שאלות טכניות מתוך כל מיני ראיונות עבודה. לא את כל השאלות זכרתי, כי בחלק מהמקרים רשמתי אותן די הרבה זמן אחרי הראיון עצמו (לכן לא כדאי להסיק מכמות ותוכן השאלות לגבי הקושי של הראיון בכל מקום ספציפי):

אינטל (חיפה)

1. יש לך מערך עם כדורים אדומים, צהובים וירוקים. אתה צריך לסדר אותו כך שכל האדומים יהיו בהתחלה וכל 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, שאליו נכנסים ויוצאים אנשים. דרך השער יכול לעבור רק בן אדם אחד בכל רגע נתון. ברשותך גלאי שמזהה מעבר של בן אדם. תכנן מערכת שתדע בכל רגע נתון כמה אנשים נמצאים בתוך האולם (מותר לך להשתמש ביותר מגלאי אחד). האם יכולה להיווצר בעיה במערכת שתיכננת?
4. נתונים Task'ים שרצים במערכת ההפעלה. כל Task מתחיל בנקודת זמן מסויימת, מסיים בנקודת זמן מסויימת, ובמהלך הריצה שלו עובר מ-Run ל-Pend ולהפך. הזמן שהוא נמצא ב-Run כל פעם הוא קבוע, והזמן שהוא נמצא ב-Pend 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 ב-Run בכל פעם שהוא עובר למצב ריצה, ומשך הזמן שהוא נמצא ב-Pend בכל פעם שהוא עובר למצב המתנה). הוסף למחלקה פונקציה שמקבלת את הזמן הנוכחי במערכת ומחזירה את מצב ה-Task שהאובייקט מתאר (Run, Pend, Not-Alive).
5. הרחב את האובייקט 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 לשימוש 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?

BATM (יוקנעם)

9. נתונות 100 מנורות בטור. כולן מכובות. 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 ישארו דולקות בסוף?
10. משחק – שני אנשים יושבים אחד מול השני, כשבאמצע שולחן עגול. לכל אחד יש מספר בלתי מוגבל של מטבעות שקל. כל אחד בתורו מניח מטבע על השולחן. מותר להניח איפה שרוצים, אבל לא על מטבעות אחרים שכבר על השולחן. הראשון שלא יכול יותר לשים מטבעות, 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 בנסיון 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. יש קומה מסויימת, 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 נסיונות.

IBM (חיפה)

13. אתה נמצא 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 שצילמת נשמרת. תאר שיטה להגיע לשומר כשבמצלמה נמצאת תמונה של בית אקראי. לרשותך פונקציה Rand. בנוסף, עליך להוכיח שעבור N בתים, ההסתברות להישאר עם תמונה של בית כלשהו היא $1/N$.
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...

מיקרוסופט (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. ממש את זה באמצעות אמצעי סנכרון מוכרים לך. לשם הפשטות, אפשר להניח שהלולאות של שניהם זהות ונקודות ההמתנה שלהם בלולאות הן גם זהות. אסור להשתמש בחוט נוסף.
18. אותה בעיה, רק עבור N חוטים.
19. כתוב פונקציה אשר מקבלת מחרוזת, ומדפיסה את כל הפרמוטציות האפשריות שלה.
20. 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?
21. נתון לך מילון. איך תייצג אותו בצורה יעילה? תאר אלגוריתם שמקבל כקלט מילה ומוצא את כל המילים במילון שהן פרמוטציה כלשהי של מילת הקלט.
22. רוצים לתכנן מעבד עם פקודות באורך קבוע – כל פקודה ביחד עם האופרנדים שלה צריכה להיות בדיוק 12 ביט. האופרנדים האפשריים הם כתובות באורך 3 ביט. סט הפקודות הנדרש כולל:
 - 4 פקודות שמקבלות 3 אופרנדים כל אחת.
 - 255 פקודות שמקבלות אופרנד אחד כל אחת.
 - 16 פקודות שלא מקבלות אופרנדים כלל.א. האם ניתן לבנות מעבד כזה? הסבר.
- ב. נסח תנאי שיבדוק האם ניתן לתכנן מעבד עבור המקרה הכללי:
 - K_1 פקודות שמקבלות P_1 אופרנדים באורך L_1, \dots
 - K_2 פקודות שמקבלות P_2 אופרנדים באורך L_2, \dots
 - K_n פקודות שמקבלות P_n אופרנדים באורך L_n .
23. נתון עץ בינארי (לכל צומת 2 בנים לכל היותר). העץ לא מאוזן ולא ממויין, ויתכנו רשומות זהות בצמתים שונים. רוצים להעביר את העץ למחשב אחר. תכנן אלגוריתם שיקודד את העץ לקובץ ואלגוריתם שיפענח את הקובץ חזרה לעץ.

KLA (מגדל העמק)

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

צורן (חיפה)

31. כתוב פונקציה שמקבלת שתי מחרוזות ומשווה ביניהן.
32. כתוב ממשק של פונקציות (קובץ Header) עבור תור FIFO.

Spectrum Dynamics (טירת הכרמל)

33. ישנו חוואי בעל פרה אחת. כל בוקר החוואי קם, הולך לנהר, לוקח מים והולך להשקות את הפרה. מצא את הדרך הקצרה ביותר שבה יכול החוואי לבצע את המשימה. ניתן להניח שהנהר הוא פס ישר ברוחב קבוע כלשהו, ושהחוואי והפרה נמצאים באותו צד שלו.
34. 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?

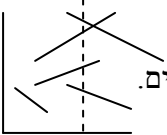
Freescape (הרצליה)

35. נתון מערך גדול מאד שמאותחל בהצהרה: $\text{int array}[\dots] = \{\dots\}$. 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?
36. בנה מערך של מספרים מ-0 עד $N-1$, שמסודרים בצורה אקראית. לרשותך פונקציה Rand שמחזירה מספר אקראי.
37. נתונה טבלה בגודל $N \times N$, 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.

שאלות ששמעתי מאחרים

38. 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 מישוה יכול לשכפל אותו בדרך ולהשתמש בו אחרי זה).
39. נסעת לכיוון אחד במהירות של 40 קמ"ש. באיזה מהירות אתה צריך לחזור ע"מ שהמהירות הממוצעת שלך (עבור כל הדרך) תהיה 80 קמ"ש?
40. נתונים שני משתנים. איך תחליף בין הערכים שלהם בלי להשתמש במשתנה נוסף?
41. נתונה טבלה של הרבה משתנים ויחסי גודל ביניהם (למשל, $a < b$, $x > y$ וכו'). תאר אלגוריתם שיאפשר לך לסדר אותם בסדר עולה.
42. נתון כלא ובו N אסירים. בחצר הכלא יש נורה ומפסק שמדליק או מכבה אותה. 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. האסירים צריכים להגיד למנהל כאשר הם בטוחים שכל האסירים כבר יצאו לחצר (מותר שחלקם יהיו בחצר יותר מפעם אחת, אבל כל אחד חייב להיות בחצר לפחות פעם אחת). בין האסירים אין שום אמצעי תקשורת, למעט המנורה כמובן. בתחילת המשימה, המנהל מאפשר לכולם לצאת לחצר בשביל לתכנן ביחד דרך פעולה. מצא שיטה שתבטיח להם הצלחה במשימה.
43. נתון מערך בגודל n . כל המספרים בתחום $[m, \dots, m+n+1]$ נמצאים במערך הזה, פרט לשניים, שהם לא המינימום או המקסימום. למשל, אם $m=2$, $n=6$, והמערך מכיל את המספרים 2,4,6,7,8,9, אז חסרים המספרים 3,5. המערך לא ממין. 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).
44. נתון שולחן עגול מסתובב עם 4 מתגי לחיצה (לא ניתן לדעת האם מתג נמצא ב-On או ב-Off). באמצע השולחן יש מנורה, אשר המתח עבודה מחובר למפסק ראשי נפרד. בהתחלה המנורה כבויה. ידוע שהמנורה הזאת דולקת 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, המנורה 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 המנורה דולקת.
45. 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.
46. 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: $| |$.
47. נתון שטח בגודל $2^N \times 2^N$ ללא המשבצת הימנית התחתונה. הוכח שניתן לרצף אותו בעזרת קבוצות של משבצות מהצורה $\begin{smallmatrix} \square & \square \\ \square & \square \end{smallmatrix}$ בלבד (מותר לסובב אותה).
48. כתוב תוכנית הבודקת כמה ביטים דלוקים יש בבית אחד. כעת כתוב אותה כך שתתבצע בזמן קבוע. מהו הטרייד אוף?

שאלות ששמעתי מאחרים (המשך)

49. 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 המחשבת את מספר האפשרויות השונות (קומבינציות שונות של קפיצות) שבהן יכול ספיידרמן 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.
50. כתוב אוטומט סופי דטרמיניסטי, שמקבל כקלט מספר בינארי מהספרה הגדולה ביותר לספרה הקטנה ביותר, ובודק אם המספר מתחלק ב-5.
51. כתוב תוכנית שמחפשת "לולאה" ברשימה מקושרת חד-כיוונית ומשתמשת בזיכרון בגודל קבוע.
52. נתון המשחק הבא: המראיין כותב שורה של מספרים כלשהם, כשהאילוף היחיד 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. מנצח מי שסכום האיברים שלקח הוא הגדול ביותר. מצא את הטריק שמבטיח ניצחון למי שמקבל את התור הראשון.
53. בהינתן מספר, איך אפשר לדעת אם הוא חזקה שלמה של 2, תוך שימוש ב- $O(1)$ פעולות חישוב בלבד?
54. איך פונקציה מחלקה יודעת איזה אובייקט קרא לה? איך נבנה קוד האסמבלי של קריאה לפונקציה כזאת ע"מ שהיא תתבצע עם האובייקט הנכון?
55. נתונה תמונה בגודל M על N . כל פיקסל מיוצג באמצעות ביט בודד. כתוב פונקציה שתחזיר את תמונת הראי שלה.
56. נתון אוסף של קטעים במערכת הצירים XY . כל קטע נתון כשתי נקודות במישור: $[(x_0, y_0), (x_1, y_1)]$. מצא את המספר הגדול ביותר של נקודות חיתוך בין ישר מאונך כלשהו לבין הקטעים. לדוגמה, בשרטוט המצורף ישנן 4 נקודות חיתוך כאלה לכל היותר.
- 
57. למה Constructor לא יכול להיות וירטואלי, ולמה Destructor צריך להיות וירטואלי?
58. פעולת הקצאה דינאמית מתבצעת בעזרת מערכת ההפעלה. כיצד ניתן למנוע את השימוש במערכת ההפעלה בעת הקצאה דינאמית של אובייקט כלשהו (למשל, $A^* a = \text{new } A$)?
59. תאר שיטה ליישם ב- $C++$ את מנגנון הפונקציות הוירטואליות ללא שימוש במילה השמורה `virtual`.
60. נתון מערך של N מספרים. מצא את המינימום והמקסימום, תוך שימוש ב- $1\frac{1}{2}N$ פעולות השוואה לכל היותר.
61. נתונה תמונה בגודל M על N . כל פיקסל מיוצג באמצעות בית אחד. תאר אלגוריתם למציאת הריבוע המקסימלי של פיקסלים מאותו צבע.
62. 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 זיכרון, שבהן תוכל התוכנית להשתמש באופן המקובל בשפת `C`:
- א. `void* MyMalloc(int size)`
- ב. `void MyFree(void* ptr)`
63. נתונה תוכנת Real-Time שמשתמשת במאגר קבוע של N חוצצים (`Buffer`'ים). התוכנה 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.
64. נתון מעבד ובו 8 רגיסטרים של 4 ביט כל אחד. ישנן 4 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.
- בצע בעזרת הפעולות הנ"ל $R3 = R1 * R2$. האם ניתן לוותר על חלק מהפעולות ולממש אותן באמצעות הפעולות האחרות? האם הקוד שכתבת עובד נכון גם עבור מספרים שליליים (כלומר, כאשר מתייחסים לערכי הרגיסטרים בשיטת המשלים ל-2)?

שאלות ששמעתי מאחרים (המשך)

65. 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 שתיצר ב- Prog ותמצא את הסיבית המנותקת הראשונה מביניהן.

66. 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?

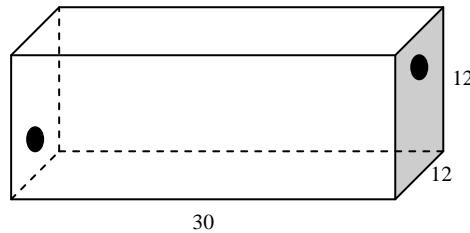
ב. אם לא, הוסף לאחד מקטעי הקוד (או לשניהם) רצף פעולות שיבטיח את זה. מה יהיה הערך של x בסיום? האם פונקצית מחלקה סטטית יכולה להיות וירטואלית? מדוע?

68. 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? ניתן להניח שמיקום הספינה ומיקום האי נתונים כנקודות במישור, ושכל קרחון מיוצג כמצולע (אוסף של קודקודים) במישור.

69. 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?

70. שלושה אנשים רוצים לדעת מהו ממוצע המשכורות שלהם. איך הם יכולים לחשב את הממוצע הזה בלי שאף אחד יגלה לאחרים מהי המשכורת שלו?

71. נתון חדר שגודלו $12 \times 12 \times 30$ מטר. הקיר השמאלי והקיר הימני הם בגודל 12×12 מטר, ושאר הקירות 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?



72. 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)?

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

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. סמלים אלה מוחלפים במהלך הקומפילציה עצמה, בכתובות שנקבעות עבורם ע"י הקומפיילר.

הקומפיילר מחלק את מרחב הזיכרון של התוכנית לשלושה אזורים בסיסיים:

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

כשהתוכנית נכנסת למצב ריצה, היא מקבלת מרחב זיכרון קבוע (Memory Address Space) שבו היא פועלת. שמות של סמלים שהוחלפו בשלב 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.

שימוש ברגיסטרים במהלך הריצה:

רגיסטר ה-PC (Program Counter) מצביע תמיד על הפקודה הבאה לביצוע, שרשומה באזור הקוד. רגיסטר ה-SP (Stack Pointer) מצביע תמיד על המקום הבא שפנוי במחסנית. רגיסטרים אחרים משמשים לצורך ביצוע פעולות חישוביות על אופרנדים. ע"מ לבצע פעולה חישובית על אופרנד כלשהו, המעבד עשוי לטעון אותו מהזיכרון לאחד מהרגיסטרים הפנויים שברשותו.

כאמור, במהלך הקומפילציה, שמות של משתנים ופונקציות מוחלפים בכתובות זיכרון. המשתנים מוחלפים בכתובות שבאזור הנתונים או במחסנית, והפונקציות מוחלפות בכתובות שבאזור הקוד.

המשתנים שמשויכים לאזור הנתונים מתחלקים לארבעה סוגים.

ההבדל ביניהם הוא אך ורק ב-Scope שבו הקומפיילר "מכיר" אותם (כלומר, בזמן ריצה אין ביניהם שום הבדל):

1. משתנים גלובליים – מוכרים בכל התוכנית.
 2. משתנים סטאטיים גלובליים – מוכרים רק בקובץ שבו הם נמצאים.
 3. משתנים סטאטיים לוקאליים – מוכרים רק בפונקציה שבה הם נמצאים.
 4. משתנים סטאטיים מחלקתיים – מוכרים רק ב-Scope המחלקה שאליה הם שייכים.
- הכתובות של משתנים אלה נשארות קבועות במשך כל ריצת התוכנית (קבועות ביחס לתחילת מרחב הזיכרון של התוכנית).

המשתנים שמשויכים למחסנית הם המשתנים הלוקאליים (אלה שמוגדרים בתוך פונקציות). משתנים אלה מוחלפים ע"י הקומפיילר בכתובות, שבזמן ריצה יהיו בעצמן יחסיות לערך של רגיסטר ה-SP. עבור פונקציה כלשהי, רגיסטר ה-SP (מצב המחסנית) עשוי להיות שונה בכל פעם שהיא נקראת. לכן גם מיקום המשתנים של הפונקציה ביחס לתחילת המחסנית עשוי להיות שונה בכל פעם שהיא רצה. כלומר, הכתובות שלהם לא נשארות קבועות (אפילו) בתוך מרחב הזיכרון של התוכנית במשך הריצה שלה.

הפונקציות גם כן מתחלקות למספר סוגים: {גלובאליות, מחלקתיות} \times {סטאטיות, לא סטאטיות}. גם במקרה הזה אין הבדל בין הסוגים השונים, בכל מה שקשור לתהליך החלפת הסמלים בכתובות. כאמור, כל סמל שמייצג פונקציה (בלי קשר לסוג שלה) מוחלף בכתובת שבאזור הקוד.

לסיכום:

כל הסמלים בתוכנית מייצגים ערכים כלשהם, שנקבעים עוד לפני ריצת התוכנית. 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 הריצה של התוכנית).

פונקציות מחלקה:

ההבדל בין פונקציות מחלקה לא סטאטיות לפונקציות מחלקה סטאטיות:

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

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

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

<u>קריאה לפונקציה וירטואלית</u>	<u>קריאה לפונקציה לא וירטואלית</u>
הפונקציה שצריכה להתבצע לא ידועה בזמן קומפילציה, כי היא תלויה בסוג האובייקט שדרכו קוראים לפונקציה	הפונקציה שצריכה להתבצע ידועה כבר בזמן קומפילציה, ולכן גם כתובתה ידועה
הקומפיילר מייצר עבור כל מחלקה טבלה, שמכילה את כתובות כל הפונקציות הווירטואליות של אותה המחלקה	
כל אובייקט מקבל את הכתובת של הטבלה של המחלקה שאליה הוא שייך, כאשר הוא נוצר (בזמן ריצה)	
עבור כל קריאה לפונקציה וירטואלית, הקומפיילר מייצר קוד של קפיצה לכתובת שרשומה בטבלה של האובייקט	עבור כל קריאה לפונקציה לא וירטואלית, הקומפיילר מייצר קוד של קפיצה לכתובת של אותה פונקציה

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
Push Ax              //Push the address of the instance into the stack
Push Cx              //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 מספר פעולות Read (קריאת תוכן המצביע של ה-V-Table) ופעולת Add אחת (בין רגיסטר למספר קבוע).

תכנות מונחה עצמים:

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

הקונספט הזה שם דגש על שני אלמנטים עיקריים:

1. Encapsulation (סגירות).

2. Derivation (גזירה).

הרעיון שעומד מאחורי Encapsulation, הוא בניית חלקים שונים בתוכנה כמודולים עצמאיים ובלתי תלויים. אפשר להסתכל על מודול בתוכנה כעל "קופסה שחורה", שמסוגלת לעשות מספר מטרות מוגדרות:

1. מה שהיא מסוגלת לעשות – היא חושפת בפני כל הצרכנים (המשתמשים).

2. איך היא עושה את זה – ידוע רק ליצרן (המממש).

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 עושה ואיך הוא עושה את זה), ואילו את שאר החלקים מספיק לו להכיר מבחוץ (מה הם ממסוגלים לעשות).

הרעיון שעומד מאחורי Derivation הוא שימוש במודולים קיימים ומוכרים לצורך הרחבתם.

ישנן שתי תבניות כלליות שבעזרתן ניתן להרחיב מודול A למודול B:

1. ליצור את B ולשים בתוכו A, בנוסף לתכונות הספציפיות שרצויות בו (במקרה הזה, B has an A).

2. לגזור את B מ-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 Stop ו-Play, ומודול B מייצג טייפ חדשני שמאפשר גם Rewind. פס הייצור של A כבר קיים, ובנוסף, קהל הצרכנים כבר מכיר את הממשק שלו. לכן, במקום ליצור את B מהתחלה, נרחיב את פס הייצור של A ונוסיף לו את מימוש הממשק של B.

כלי נוסף שטמון ב-Derivation, 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 שאז ערכם יורד.

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

לסיכום, Derivation מאפשר 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 ב-DDL'ים.

ניהול שכבות הזיכרון:

שכבת הזיכרון הקרובה ביותר למעבד היא הרגיסטרים. המעבד יכול לבצע את כל הפעולות החישוביות שלו על שכבה זו. המעבד יכול לאחסן בה מידע רק כל עוד מתקבלת אספקת מתח סדירה.

מעליה נמצאת שכבת הזיכרון הפנימי.

המעבד יכול לבצע את חלק מהפעולות החישוביות שלו גם על שכבה זו. המעבד יכול לאחסן בה מידע רק כל עוד מתקבלת אספקת מתח סדירה. בדרך כלל, שכבה זו מחולקת לשתי רמות:

1. Cache – זיכרון שנמצא יותר קרוב למעבד, ולכן הגישה אליו יותר מהירה. שכבה זו נקראת לעיתים גם L2.
2. Ram – זיכרון שנמצא פחות קרוב למעבד, ולכן הגישה אליו יותר איטית. שכבה זו נקראת לעיתים גם L1.

מעליה נמצאת שכבת הזיכרון המשני (Hard Disk).

המעבד לא יכול לבצע את הפעולות החישוביות שלו על שכבה זו. המעבד יכול לאחסן בה מידע שנשמר גם כאשר מפסיקה אספקת מתח סדירה.

על מנת שרצף פעולות (קטע קוד) כלשהו ירוץ, הוא חייב להיטען לזיכרון הפנימי. המונח זיכרון וירטואלי מתאר צורת ניהול מתוחכמת של שכבות הזיכרון השונות. מיפוי כתובות הזיכרון הווירטואלי לכתובות הזיכרון הפיזי מתבצע ע"י 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 – ע"מ שקטע קוד כלשהו ירוץ, הדבר היחידי שחייב להיות בזיכרון הוא הפעולה הבאה לביצוע:

- כאשר יש צורך בנתון כלשהו, מחפשים אותו ב-Cache.
 - אם הנתון לא נמצא שם (Miss), מחפשים אותו ב-Ram.
 - אם הנתון לא נמצא שם (Page Fault), מחפשים אותו ב-Hard Disk.
- בכל שלב, במידה והנתון לא נמצא במקום המתאים, ה-MMU "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, נתונים מוצאים באופן זמני לזיכרון המשני, 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, המעבד 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 וחוסים:

תהליך (Process) הוא למעשה תוכנית בביצוע. הגנה על תהליך מפני תהליכים אחרים מתבצעת כאמור בעזרת ה-MMU, שמנהלת את הזיכרון הווירטואלי. עבור כל תהליך יש רגיסטרים, שבעזרתם מוגדר מרחב הזיכרון הפיזי שלו. בכל פעולת גישה של התהליך לזיכרון, כתובת היעד מחושבת באמצעות הרגיסטרים האלה וטבלת הזיכרון הווירטואלי, כך שלמעשה אף תהליך לא יכול לגשת למרחב הזיכרון הפיזי של 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 הנוכחי של מרחב הזיכרון של התהליך ואת המצב הנוכחי של הרגיסטרים.

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

כאמור, אין צורך לבנות תוכנית, כך שהתהליך שמייצג אותה יהיה מוגן מפני תהליכים אחרים – מערכת ההפעלה דואגת לזה. לעומת זאת, בבניית תוכנית שיוצרת חוסים, 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 חוסים מסויימים מריצים, ובמהלכם הם ניגשים לאותו אזור בזיכרון המשותף. אם אחד מהחוסים 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 לאחרים לבצע את הקטע הקריטי שלהם. לצורך כך, יש לממש מנגנון שיבטיח מניעה הדדית (Mutual Exclusion). המנגנון שממומש, חייב גם למנוע מצב שבו קבוצה של חוסים נמצאת בהמתנה "ציקלית" לכניסה של אחד מהם לקטע הקריטי (Deadlock). בנוסף, הוא גם חייב למנוע מצב שבו חוט כלשהו ממתין לנצח להתחיל את הקטע הקריטי שלו (Starvation).

שיטות לסנכרון בין חוסים:

1. Busy Wait (Polling) – נגדיר משתנה X באזור הזיכרון המשותף (משתנה גלובאלי), ונאתחל אותו ל-1. כל חוט, לפני כניסה לקטע הקריטי, ימתין עד שהערך של X הוא 1. בתחילת הקטע הקריטי, החוט ישנה את X ל-0, וביציאה ממנו הוא ישנה אותו בחזרה 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.
2. Semaphore – נגדיר אובייקט Sem שמאפשר שתי פעולות אטומיות (פעולה אטומית היא רצף של פעולות, אשר ידוע בוודאות שבמהלכו לא מתרחשת החלפת הקשר):
 - Sem.Wait() – אם Sem.counter גדול מאפס, הקטן אותו ב-1. אחרת, הוצא את החוט הנוכחי להמתנה.
 - Sem.Signal() – אם קיים חוט שנמצא בהמתנה, הוצא אותו ממנה. אחרת, הגדל את Sem.counter ב-1.
3. Monitor – נגדיר אובייקט Mon אשר מחזיק את "הנתונים הקריטיים" של התוכנית (אלה שמספר חוסים עשויים לגשת אליהם ולפחות אחד מהחוסים עשוי לשנות אותם). כל גישה של אחד החוסים לנתונים הקריטיים תתבצע אך ורק דרך ממשק הפונקציות של האובייקט. אותן פונקציות יאפשרו בכל רגע נתון רק לחוט אחד להשתמש בהן. ה-Monitor למעשה עובד על אותו עיקרון כמו ה-Semaphore, אבל מספק ממשק יותר נוח לצורך יישום הסנכרון בין חוסים (שכאמור דורש, Mutual Exclusion, מניעת Deadlock ומניעת Starvation).

תשובה לשאלה 1

שלב ראשון – נעביר את כל הכדורים האדומים לתחילת המערך:
א. נחזיק אינדקס A שמצביע על תחילת המערך ואינדקס B שמצביע על סוף המערך.
ב. נקדם את אינדקס A לכיוון סוף המערך, עד שניתקל בכדור שאינו אדום.
ג. נקדם את אינדקס B לכיוון תחילת המערך, עד שניתקל בכדור אדום.
ד. נחליף בין הכדורים שמוצבעים ע"י האינדקסים.
ה. נחזור על שלבים ב', ג', ו-ד', עד שאינדקס A יעבור את אינדקס B.
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 גלמים שמונחים בצורה טורית: גלאי A לפני הכניסה לאולם וגלאי B אחרי הכניסה לאולם. עבור כל גלאי נחזיק משתנה בוליאני f , שיאותחל ל-0. בנוסף, נחזיק משתנה שמונה את 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 מישו משנה את דעתו באמצע הדרך (בין הגלאים) וחוזר.

תשובה לשאלה 4

אם הזמן הנוכחי לא נמצא בין זמן ההתחלה לזמן הסיום של ה-Task, אז ה-Task לא קיים. אם הזמן הנוכחי כן נמצא בין זמן ההתחלה לזמן הסיום של ה-Task:

Begin Time	Current Time				End Time
Running	Pending	Running	Pending	Running	Pending

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

STATE Task::GetState(iCurrentTime)

```
{
    if (m_iBeginTime<=iCurrentTime && iCurrentTime<=m_iEndTime)
    {
        int iModulo=(iCurrentTime-m_iBeginTime)%(m_iRunTime+m_iPendTime);
        if (iModulo<iRunTime)
            return RUNNING;
        else
            return PENDING;
    }
    else
        return NOT_ALIVE;
}
```

ההנחה היא כמובן, שה-Task תמיד מתחיל ב-Running. אם הוא מתחיל ב-Pending, אז צריך להחליף את התנאי $iModulo < iRunTime$ בתנאי $iModulo \geq iPendTime$.

תשובה לשאלה 5

כדי לדעת את מספר האובייקטים מסוג מסוים 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

משתנה סטאטי ב-C יכול להופיע ברמת הפונקציה (משתנה לוקאלי) וברמת הקובץ (משתנה גלובאלי). ההבדל הוא רק ב-Scope שבו הקומפילר מזהה אותו. בכל מקרה, המשתנה ימוקם ב-Data Section. משתנה סטאטי ב-C++ יכול להופיע גם ברמת המחלקה. גם כאן ההבדל הוא רק ב-Scope שבו הקומפילר מזהה אותו, וגם כאן המשתנה ימוקם ב-Data Section (ולא יגדיל את שטח הזיכרון שאובייקט מהסוג של המחלקה תופס). פונקציה סטאטית ב-C יכולה להופיע ברמת הקובץ, וזה רק אומר שהקומפילר לא יכיר אותה בקבצים אחרים. פונקציה סטאטית ב-C++ יכולה להופיע גם ברמת המחלקה (פונקציה מחלקתית), והיא למעשה כמו פונקציה גלובאלית, למעט העובדה שאפשר לפנות אליה רק דרך המחלקה – `MyClass::Func(...)`. בניגוד לפונקציות מחלקה שאינן סטאטיות, היא לא מקבלת מצביע לאובייקט מסוג המחלקה (אין לה "this"), ולכן לא יכולה לגשת לשדות לא סטאטיים או לפונקציות לא סטאטיות של המחלקה בלי שיש ברשותה אובייקט מתאים. לפירוט נוסף – חלק 3 (חומר כללי: סמלים בתוכנה).

תשובה לשאלה 7

נבצע N איטרציות.

באיטרציה מספר i:

- ברשימת הקלט, נבחר שיר אקראי מתוך N-i השירים הראשונים.
- ברשימת הפלט, נכניס אותו למקום ה-i.
- 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:
- נשתמש בפונקציה Rand ע"מ לבחור מספר אקראי מהרשימה החלקית שחישבנו. נקרא לו P.
 - נתחיל משיר מספר P (ברשימת השירים המלאה), ונתקדם P שירים בכל איטרציה, במשך N איטרציות.
 - 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:
- זמן חישוב הרשימה החלקית עלול להיות ארוך (למרות שזה לא עקרוני, כי מחשבים אותה רק פעם אחת).
 - 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 שאנחנו יכולים לקבל בשיטה הזאת = גודל הרשימה החלקית $N > N$.
 - מספר הסידורים השונים האפשריים עבור N שירים $N = 1 * 2 * \dots * N$ עצרת.

תשובה לשאלה 8

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

- לשים את ה-Constructor ב-protected של המחלקה. ניתן יהיה ליצור עצמים של מחלקות שיורשות ממנה.
- לשים את ה-Constructor ב-private של המחלקה, ולהוסיף פונקציה סטאטית שיוצרת עצם מהסוג שלה ומחזירה מצביע אליו (בשיטת ה-Singleton).

תשובה לשאלה 9

המנורות שיישאר דולקות: אלה שמספר המחלקים של האינדקס שלהן הוא אי זוגי. האינדקסים שמספר המחלקים שלהם הוא אי זוגי: מספרים ריבועיים – 1, 4, 9, 16, 25...

תשובה לשאלה 10

אם לא ידוע מי מתחיל, אז אין שיטה כזאת, כי זה פרדוקס (אם הייתה שיטה כזאת, 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:

$$A = \text{ההסתברות לגשת לסל הראשון} = 1/2$$

$$B = \text{ההסתברות להוציא כדור אדום מהסל הראשון} = 1/1$$

$$C = \text{ההסתברות לגשת לסל השני} = 1/2$$

$$D = \text{ההסתברות להוציא כדור אדום מהסל השני} = 9/19$$

נקבל את התוצאה:

$$A * B + C * D = \text{ההסתברות להוציא כדור אדום} = 14/19$$

תשובה לשאלה 12

את הכדור הראשון נזרוק מקומה 14.

אם הוא נשבר, ננסה את הכדור השני מקומה 1 עד לקומה 13 (או עד שהוא יישבר).

סה"כ במקרה הגרוע – 14 ניסיונות (14, 13-1).

אם הכדור הראשון לא נשבר, נזרוק אותו מקומה 27.

אם הוא נשבר, ננסה את הכדור השני מקומה 15 עד לקומה 26 (או עד שהוא יישבר).

סה"כ במקרה הגרוע – 14 ניסיונות (14, 27, 26-15).

נמשיך באותו אופן, כשבכל פעם אנחנו עולים קומה אחת פחות ממה שעלינו בפעם הקודמת (14, 27, 39, 50, 60, 69,

77, 84, 90, 95, 99) ולכן מספר הניסיונות במקרה הגרוע יישאר 14.

אם בקומה 99 (לאחר 11 ניסיונות) הכדור הראשון עדיין לא נשבר, נזרוק אותו מקומה 100. סה"כ – 12 ניסיונות.

לסיכום, לכל היותר נבצע 14 ניסיונות.

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 $\text{Rand}()\%2$.

את בית מספר 3 נצלם בהסתברות $1/3$, תוך שימוש בפונקציה $\text{Rand}()\%3$.

את בית מספר X נצלם בהסתברות $1/X$, תוך שימוש בפונקציה $\text{Rand}()\%X$.

נוכיח באינדוקציה, שעבור כל מספר N של בתים, ההסתברות לקבל תמונה של בית כלשהו היא שווה (וערכה הוא $1/N$):

1. עבור $N=1$, בית מספר 1 מתקבל בהסתברות של 100%.

2. עבור $N=2$, בית מספר 2 מתקבל בהסתברות של 50%, ולכן ההסתברות של בית מספר 1 יורדת ל-50%.

3. נניח נכונות עבור $N=k$ ונוכיח עבור $N=k+1$:

- לפי הנחת האינדוקציה, ההסתברות לתמונה של כל אחד מ- k הבתים היא שווה (וערכה הוא $1/k$).
- את בית מספר $k+1$ נצלם בהסתברות $1/(k+1)$, ולכן ההסתברות שתישאר בידנו תמונה של אחד מ- k הבתים הקודמים היא $1-1/(k+1)$, כלומר $k/(k+1)$.
- מכיוון שההסתברות לתמונה של כל אחד מ- k הבתים היא עדיין שווה, נקבל שההסתברות לתמונה של כל בית מתוך k הבתים היא $[k/(k+1)]/k$, כלומר $1/(k+1)$.
- מסקנה – ההסתברות של בית מספר $k+1$ שווה להסתברות של כל בית מתוך k הבתים: $1/(k+1)$.

תשובה לשאלה 14

אותו עקרון כמו בתשובה לשאלה 7.

תשובה לשאלה 15

פונקציות וירטואליות זהו הכלי שבעזרתו (ובעזרת הורשה) ניתן לייצר Polymorphism ב-C++. הפונקציה שצריכה להתבצע לא ידועה בזמן קומפילציה, כי היא תלויה בסוג האובייקט שדרכו קוראים לפונקציה. הקומפיילר מייצר עבור כל מחלקה טבלה, שמכילה את כתובות הפונקציות הווירטואליות של אותה המחלקה. כל אובייקט מקבל את הכתובת של הטבלה של המחלקה שאליה הוא שייך, כאשר הוא נוצר (בזמן ריצה). עבור כל קריאה לפונקציה וירטואלית, הקומפיילר מייצר קוד של קפיצה לכתובת שרשומה בטבלה של האובייקט. לפירוט נוסף – חלק 3 (חומר כללי: פונקציות מחלקה).

תשובה לשאלה 16

שימושי ב-C (ב-C++), אפשר להשתמש בערכי Default עבור הארגומנטים שהפונקציה מקבלת).

החתימה של הפונקציה נראית כך: `void func(int x, ...)`.

הערך המוחזר יכול להיות מכל סוג שהוא, וחייב להיות לפחות ארגומנט אחד מוצהר.

לאחר הארגומנטים המוצהרים באות שלוש נקודות (...), שמעידות על מספר משתנה של ארגומנטים.

הארגומנטים הלא מוצהרים מועברים לפונקציה, כמו שאר הארגומנטים, במחסנית. ע"מ לגשת אליהם:

- 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");
```

נממש 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 = "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 מסיכרון" לגביו ולגבי כל שאר הארגומנטים שבאים אחריו (הגודל יכול להיות 4 בתים או 8 בתים).
דוגמאות:

1. הרצה של `printf("%d %d",5)`:

- הפונקציה ניגשת לאזור שלא הוגדר עבורה בזיכרון וקוראת "זבל" (או מפילה את התוכנית).

2. 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

נגדיר משתנה גלובלי x , נאתחל אותו ל-0 ונבצע עליו Busy Wait בכל חוט באופן הבא:

```
if (x==0)
{
    x=1;
    while (x==1) { }
}
else
    x=0;
```

החוט הראשון שיגיע ללולאה בכל פעם, יגלה ש- x שווה ל-0, ישנה אותו ל-1 וימתין עד שהוא יחזור להיות 0. החוט השני שיגיע ללולאה בכל פעם, יגלה ש- x שווה ל-1 וישנה אותו ל-0. בשלב זה, שני החוטים ימשיכו לרוץ. הערות:

1. השימוש ב-Busy Wait הוא בזבזני מבחינת זמן, מכיוון שבזמן שהחוט הראשון שהגיע ללולאה ממתיין שהחוט השני יגיע אליה, עלולות להתבצע החלפות הקשר רק לצורך בדיקת הערך של x (בקוד של החוט הראשון).
2. סדרת הפעולות [בדיקת הערך של x , שינוי הערך של x] חייבת להיות מוגדרת כפעולה אטומית אחת, אשר במהלכה לא יכולה להתבצע החלפת הקשר.

תשובה לשאלה 19

נסמן מחרוזת באורך של n תווים באופן הבא: $S[1 \dots n]$.

נממש פונקציה רקורסיבית, שמקבלת מחרוזת $S[1 \dots n]$ 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 $S[x+1 \dots n]$, 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 List(1);
    List prevList = Permute(SubString(string,1,string.length));
    List nextList = List(string.length*prevList.length);
    for (int m=0; m<prevList.length; m++)
    {
        for (int n=0; n<string.length; n++)
        {
            nextList[m*string.length+n] += SubString(prevList[m],0,n);
            nextList[m*string.length+n] += string[0];
            nextList[m*string.length+n] += SubString(prevList[m],n,string.length-1);
        }
    }
    return nextList;
}
```

תשובה לשאלה 20

ניעזר במערך בוליאני שגודלו 256 תאים (כמספר התווים בקוד ASCII).
נעבור על אוסף התווים Set, ועבור כל תו נרשום true במקום המתאים לו במערך.
נעבור על המחרוזת String, ונחפש את התו הראשון אשר במקום המתאים לו במערך רשום true.
ברגע שנמצא תו כזה, נחזיר את האינדקס שלו.

```
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, נצטרך לממש טבלה שמאפשרת Hashing בעזרת מספר קבוע של פעולות (ללא 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 21

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 השנייה 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 שנמש.
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 מסויים שיוגדר מראש).

ע"מ למצוא במילון את כל המילים שהן פרמוטציות של מילה מסוימת, ישנן שתי אפשרויות:

1. נשתמש בפונקציה שתוארה בתשובה לשאלה 19, 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.
2. 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 22

ישנם סה"כ 4096 רצפים אפשריים: $2^{12} = 4096$. כאמור, כל אופרנד הוא באורך 3 ביטים.
עבור 4 פקודות שמקבלות 3 אופרנדים, דרושים לנו 2048 רצפים שונים: $4 * 2^{(3*3)} = 2048$.
עבור 255 פקודות שמקבלות אופרנד אחד, דרושים לנו 2040 רצפים שונים: $255 * 2^{(1*3)} = 2040$.
עבור 16 פקודות 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 לבנות מעבד כזה.
תנאי עבור המקרה הכללי: הסכום $\sum [K_i * 2^{(P_i * L_i)}]$ צריך להיות קטן ממספר הרצפים האפשריים.

תשובה לשאלה 23

קידוד העץ לקובץ:

1. נעבור על העץ בשיטת Pre-Order (אב, בן שמאלי, בן ימני), ולכל צומת ניתן מספר סידורי עולה.
 2. נרשום בקובץ את מספר הצמתים שיש בעץ.
 3. נקצה מערך של מצביעים ונעבור על העץ שוב.
 4. לכל צומת, נרשום בתא שמספרו הוא המספר הסידורי של הצומת את הכתובת של הצומת.
 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 24

מחלקה אבסטרקטית זאת מחלקה שיש בה פונקציה וירטואלית טהורה, ולכן אי אפשר ליצור עצמים מהסוג שלה. מחלקה כזאת משמשת כממשק גנרי, אשר מחלקות שנגזרות (יורשות) ממנה צריכות לממש.

תשובה לשאלה 25

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 26

אם ה-Copy Constructor של מחלקה כלשהי לא מוגדר באופן מפורש ע"י המתכנת, אז הקומפיילר מגדיר אותו כברירת מחדל. במקרה כזה, כאשר ה-Copy Constructor של עצם אחד מקבל עצם אחר, הוא פשוט מעתיק את השדות שלו. אם חלק מהשדות האלה הם מצביעים לזיכרון כלשהו שהוקצה בצורה דינאמית, אז שני העצמים מתייחסים לאותו אזור בזיכרון ועלולים לבצע עליו את אותן פעולות (כולל שחרור הקצאה).

תשובה לשאלה 27

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 רשומה בקוד קריאה לפונקציה וירטואלית באמצעות מצביע, הקומפיילר לא יודע איזה פונקציה בשרשרת ההורשה צריכה להתבצע, מכיוון שהוא יודע לזהות את סוג המדויק של האובייקט רק אם מדובר בעצם "ממשי" (ולא במצביע). במקרה כזה הוא 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 נועדה ע"מ לאפשר קריאה לפונקציות הווירטואליות האלה באמצעות מצביע.
 - כמו כן, הקומפיילר מוסיף לרשימת השדות של כל מחלקה מצביע ל-V-Table.
 - עבור כל קריאה לפונקציה וירטואלית באמצעות מצביע, הקומפיילר מקודד מספר פקודות מכונה, שבסופן קפיצה לכתובת שרשומה ב-V-Table של האובייקט, שבאמצעותו נקראה הפונקציה.
 - כל אובייקט מקבל את הכתובת של ה-V-Table המתאים לסוג שלו, רק כאשר הוא נוצר (בזמן ריצה).
- לפירוט נוסף – חלק 3 (חומר כללי: פונקציות מחלקה).

תשובה לשאלה 28

Singleton זה Design Pattern של מחלקה, שמאפשר ליצור רק עצם יחיד מסוגה בכל המערכת:

- נגדיר מחלקה בשם MyClass, ונשים את ה-Constructor שלה ב-private.
- כך אי אפשר ליצור Instance שלה בצורה מפורשת, וגם לא לרשת אותה (וליצור עצם שנגזר ממנה).
- נגדיר בתוך המחלקה משתנה סטאטי שיצביע לעצם היחיד מהסוג שלה, MyClass* m_pInstance.
- נאתחל אותו: MyClass* MyClass::m_pInstance = NULL.
- נגדיר בתוך המחלקה פונקציה סטאטית שתיצור עצם חדש רק אם המצביע הוא עדיין NULL.
- הפונקציה תיצור עצם חדש רק פעם אחת ותחזיר את המצביע תמיד.

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

תשובה לשאלה 29

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 נכנס 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 30

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 את התקשורת בין החוטים באמצעות אובייקטים גלובאליים (משתנים, סמאפורים וכו'), אשר לכל החוטים יש גישה אליהם.
- לפירוט נוסף – חלק 3 (חומר כללי: ניהול תהליכים וחוטים).

תשובה לשאלה 31

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

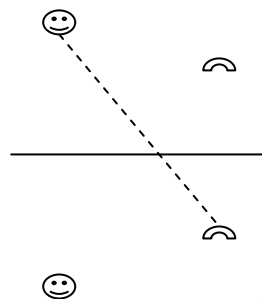
תשובה לשאלה 32

```
void InitQueue(Queue* pQueue, int iSize);
bool Insert(Queue* pQueue, const Item* pItem);
bool Extract(Queue* pQueue, Item** pItem);
bool GetFirst(const Queue* pQueue, Item** pItem);
bool IsFull(const Queue* pQueue);
bool IsEmpty(const Queue* pQueue);
```

תשובה לשאלה 33

נניח שהנהר, החוואי והפרה ממוקמים באופן הבא:

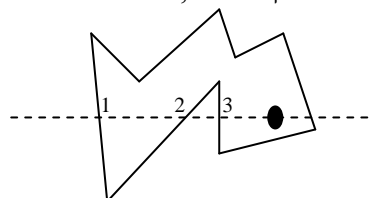
- גדת הנהר שקרובה לחוואי ולפרה – על ציר ה-X.
 - החוואי – בנקודה $(0,a)$, שנמצאת בחלק החיובי של ציר ה-Y.
 - הפרה – בנקודה (b,c) , שנמצאת ברבע החיובי של מערכת הצירים XY.
- החוואי והפרה ממוקמים בנקודות קבועות. החוואי צריך ללכת אל הפרה דרך הנקודה $(x,0)$, שנמצאת על גדת הנהר. לכן, אפשר לתאר את המרחק שהחוואי צריך ללכת כפונקציה של הנקודה הנ"ל: $f(x) = \sqrt{(x^2+a^2)} + \sqrt{((b-x)^2+c^2)}$. 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 צריך החוואי ללכת.

תשובה לשאלה 34

נתונה צורה כאוסף של קודקודים, ונקודת קלט כלשהי. נחשב את המשוואות של כל אחת מצלעות הצורה. נבדוק אם היא קיימת צלע, אשר נקודת הקלט מקיימת את המשוואה שלה וגם נמצאת בין שני הקודקודים שלה. אם קיימת צלע כזאת, אז נקודת הקלט נמצאת על השפה של הצורה. אחרת, היא נמצאת בתוך הצורה או מחוץ לצורה. נעביר דרך הנקודה ישר שמקביל לציר ה-X. נבחר נקודה על הישר שנמצאת מחוץ לצורה (נקודה שקואורדינטת ה-X שלה קטנה מקואורדינטות ה-X של כל קודקודי הצורה). נסרוק את הישר החל מאותה נקודה ועד לנקודת הקלט, ונספור כמה פעמים הוא חותך את צלעות הצורה (נתעלם מצלעות שנמצאות בדיוק על הישר). אם ספרנו מספר אי זוגי של פעמים, אז נקודת הקלט נמצאת בתוך הצורה, ואם ספרנו מספר זוגי של פעמים, אז נקודת הקלט נמצאת מחוץ לצורה. לדוגמא, בשרטוט הבא נקודת הקלט נמצאת בתוך הצורה, כי הישר שעובר דרכה חותך את הצורה 3 פעמים:



תשובה לשאלה 35

רשימת הערכים שאיתה מאתחלים את המערך תהיה חלק מקוד התכנית 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 36

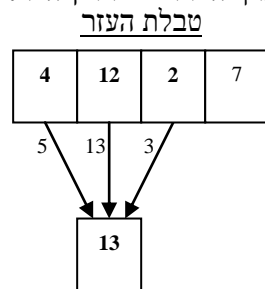
נמלא מערך במספרים 0 עד $N-1$ (לפי הסדר), ונבצע N איטרציות. באיטרציה מספר i נחשב מספר אקראי j , ונחליף בין האיבר שיושב במקום ה- i לאיבר שיושב במקום ה- j :

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

תשובה לשאלה 37

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

- נתחיל בשורה הראשונה, ובכל משבצת פשוט נרשום את הערך שרשום במשבצת שמקבילה לה.
- 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 השנייה בשורה השנייה נחשב את הערך המתאים באופן הבא:



- 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 38

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

תשובה לשאלה 39

נסמן:

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

ידוע לנו:

- $T_1 = \frac{1}{2}S / 40$ (נתון)
- $V = S / (T_1 + T_2)$ (מהירות שווה מרחק חלקי זמן)

ולכן נובע:

$$T_2 = S / V - \frac{1}{2}S / 40$$

אם נרצה שיתקיים:

$$V = 80$$

אז נצטרך שיתקיים:

$$T_2 = S / 80 - \frac{1}{2}S / 40 = 0$$

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

תשובה לשאלה 40

נניח שמספר אחד נתון ברגיסטר A והמספר השני נתון ברגיסטר B. נבצע את פעולות ההשמה הבאות לפי הסדר:

$$A = A + B$$

$$B = A - B$$

$$A = A - B$$

תשובה לשאלה 41

- 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, כי לא יכול להיות מצב שבו $X < Y < Z < \dots < X$. בהנחה שלכל זוג משתנים נתון בטבלה היחס ביניהם, נקבל גרף של רכיב אחד (ולא מספר רכיבים בלתי קשירים). נתחיל מהצומת שמייצג את המשתנה הכי קטן, ונבצע על הגרף מיון טופולוגי:
- חיפוש לעומק (DFS) שבמהלכו, בכל כניסה לצומת רושמים בצומת את זמן הכניסה.
 - סידור הצמתים בסדר עולה לפי זמן הכניסה שנרשם בהם במהלך החיפוש.

תשובה לשאלה 42

- האסירים מחלקים לעצמם מספרים בין 1 ל-N, וקובעים ביניהם מה כל אסיר צריך לעשות כשהוא יוצא החוצה. בנוסף, לפני שהמנהל מחזיר אותם לתאים, הם דואגים שהנורה תהיה כבויה. כל אסיר, חוץ מאסיר מספר N, מדליק את הנורה אם היא כבויה ובתנאי שהוא עדיין לא הדליק אותה אף פעם. אסיר מספר N מכבה את הנורה אם היא דולקת, וסופר את מספר הפעמים שהוא כיבה אותה. לאחר N-1 פעמים, הוא מודיע למנהל שכל האסירים כבר יצאו לחצר.

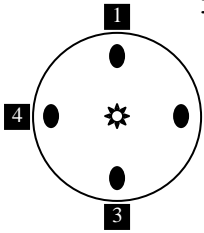
תשובה לשאלה 43

- 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 האיברים שלו ואת מכפלת האיברים שלו (נסמן S_0 ו- P_0 בהתאמה).
 2. מכיוון ש-m ו-n ידועים, ניתן לחשב את הסכום ואת המכפלה של כל המספרים בתחום $[m, \dots, m+n+1]$.
 3. נחשב את סכום שני האיברים שחסרים ואת מכפלת שני האיברים שחסרים (נסמן S ו-P בהתאמה):

$$P = \prod [m, \dots, m+n+1] - P_0, \quad S = \sum [m, \dots, m+n+1] - S_0$$
 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 44

מכיוון שיש 4 מתגים שכל אחד מהם נמצא באחד מ-2 מצבים (On או Off), ישנם סה"כ 16 מצבים אפשריים לשולחן כולו. חלק מהמצבים הם זהים לאחרים – מספיק לסובב את השולחן בלי לגעת באף מתג בשביל לעבור ביניהם, אבל בגלל שאנחנו יושבים במקום קבוע ליד השולחן (ולא מסתובבים ביחד איתו), המצבים האלה יראו לנו שונים. בכל פעם שהשולחן יסתובב, אנחנו נשב מול מתג אחר ולא נוכל לדעת מה מצב המתגים שמולנו, לכן:



- בכל שלב אי-זוגי, לאחר סיבוב השולחן, נלחץ על המתג שמיימנו ועל המתג שלידנו.
- בכל שלב זוגי, לאחר סיבוב השולחן, נלחץ על המתג שמיימנו ועל המתג שמשמאלנו.
- לדוגמא, נניח שאנחנו יושבים בכסא מספר 2:
- פעם אחת נלחץ על המתג שנמצא ליד כסא מספר 1 ועל המתג שנמצא ליד כסא מספר 2.
- פעם שניה נלחץ על המתג שנמצא ליד כסא מספר 1 ועל המתג שנמצא ליד כסא מספר 3.

בשיטה הזאת מובטח לנו, שבכל שלב לפחות מתג אחד ישנה את מצבו. על מנת להוכיח, שמכל מצב התחלתי של המתגים מגיעים למצב שבו כולם ב-On או כולם ב-Off, צריך להראות את כל המסלולים האפשריים עבור כל מקרה. מכיוון שישנם 16 מקרים התחלתיים אפשריים, ולאחר כל שלב ישנן 4 אפשרויות שונות לסיבוב השולחן, ההוכחה קצת ארוכה. שתי דרכים שונות לרשום אותה:

א. נצייר גרף יחיד, שבו נייצג את המצבים כצמתים, ואת המעברים בין המצבים כקשתות בין הצמתים.

ב. נמלא שתי טבלאות, שבהן נייצג כל מצב ע"י מספר בינארי בן 4 ספרות:

- טבלה א': בכל שורה נרשום מצב כלשהו, ומולו את 4 המצבים הבאים האפשריים לאחר שלב אי-זוגי.
- טבלה ב': בכל שורה נרשום מצב כלשהו, ומולו את 4 המצבים הבאים האפשריים לאחר שלב זוגי.

תשובה לשאלה 45

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.

עבור המקרה הכללי, נשתמש ב-i ו-j כאינדקסים לטבלה של 9×9 משבצות (טווח הערכים שלהם יתחיל מ-0). נקבל את הנוסחא:

$$\text{Table}[i][j] = ((i * 6 - i / 3 + j) \% 9) + 1$$

תשובה לשאלה 46

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

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

בשביל 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 ונסיים את סריקת הביטוי עד סופו, נבדוק אם המחסנית ריקה. אם לא, נכריז שהוא לא חוקי. אחרת, נדע שהוא חוקי.

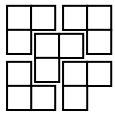
אם קיים סוג רביעי של סוגריים, שבו אין הבדל בין סוגר שמאלי לסוגר ימני: |, לא נוכל לדעת אם לבצע Push או Pop כאשר ניתקל בסוגר מהסוג הזה. לכן, נוציא את האיבר האחרון שהכנסנו למחסנית (פעולת Pop) ונבדוק אם הוא מאותו סוג. אם לא, נכניס אותו בחזרה למחסנית, ובנוסף נכניס גם את הסוגר שניתקלנו בו למחסנית.

תשובה לשאלה 47

עבור $N=1$, נקבל שטח בגודל 2×2 ללא המשבצת הימנית התחתונה, וניתן לרצף אותו באמצעות הצורה:

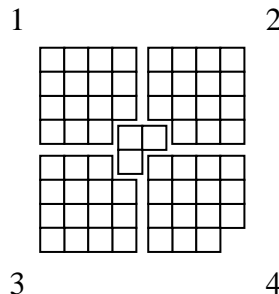


עבור $N=2$, נקבל שטח בגודל 4×4 ללא המשבצת הימנית התחתונה, וניתן לרצף אותו באמצעות הצורות:



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

- לפי ההנחה, ניתן לרצף שטח בגודל $2^k \times 2^k$ 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, ניתן למלא באמצעות הצורה:
 - ביחד עם החלק הרביעי, נקבל שטח בגודל $2^{(k+1)} \times 2^{(k+1)}$ ללא המשבצת הימנית התחתונה.
- לדוגמא, נבנה באמצעות השטח שקיבלנו עבור $N=2$, שטח עבור $N=3$:



חישוב מספר הצורות שבהן נשתמש בשביל לרצף שטח בגודל $2^n \times 2^n$ ללא המשבצת הימנית התחתונה:

$$\begin{aligned} A_1 &= 1 \\ A_{n+1} &= 4A_n + 1 \end{aligned}$$

תשובה לשאלה 48

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

על מנת שהפונקציה תתבצע בזמן קבוע $O(1)$, אפשר להכין מראש Hash Table עבור כל הערכים האפשריים של בית אחד. הטרייד אוף יהיה בצריכת הזיכרון.

תשובה לשאלה 49

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

קריאה ל- $\text{Count}(n)$ תחזיר את מספר האפשרויות השונות לטיפוס על בניין בן n קומות. 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 (תכנון דינאמי):

```
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 50

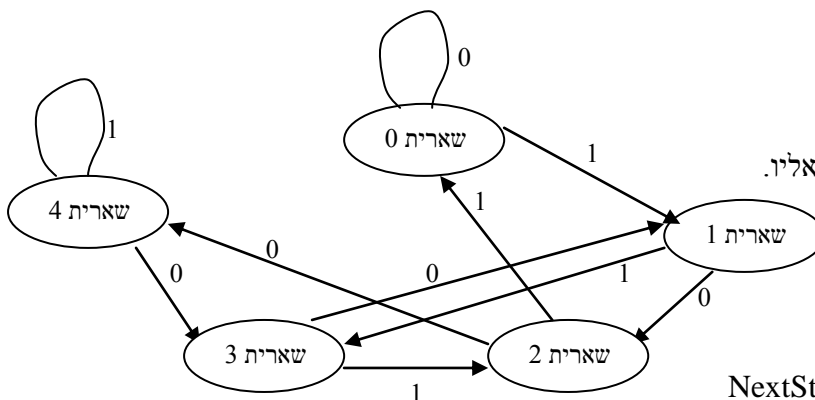
סה"כ 5 מצבים.

מצב "שארית 0" הוא המצב ההתחלתי.

מצב "שארית 0" הוא גם ה-ACCEPT.

חמישית מהחיצים (2 מתוך 10) מגיעים אליו.

כל שאר המצבים שקולים ל-REJECT.



עבור כל קלט (0 או 1), מתבצע:

$$\text{NextState} = (2 * \text{CurrState} + \text{Input}) \% 5$$

תשובה לשאלה 51

- 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 N כלשהו (בהמשך הרשימה), ונחפש אותם באופן הבא:
1. לכל צומת בקטע הצמתים שהגדרנו, נבדוק אם הקשת מצומת N מובילה אליו.
 2. אם נמצא צומת כזה, נדע בוודאות שקיימת "לולאה" ברשימה.
 3. אם לא, נבדוק אם הקשת מצומת N, 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 N 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 $O(1)$.

תשובה לשאלה 52 (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 יותר.
- 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 53

- 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 שני הרצפים תיתן אפס, אם ורק אם A הוא חזקה שלמה של שתיים: $A \& (A-1) = 0$.

תשובה לשאלה 54

פונקציה 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 55

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

א. נכין Look-Up Table עבור כל הערכים האפשריים שבית יכול לקבל (בין 0 ל-255), כך שכל בית ימופה לבית ההפוך לו (מבחינת ערכי הביטים שלו):

```
void CreateLut(unsigned char lut[256])
{
    for (int i=0; i<256; i++)
    {
        lut[i] = 0;
        for (int j=0; j<8; j++)
            lut[i] |= ((i>>j)&1)<<(8-1-j);
    }
}
```

ב. נעבור על כל שורה בתמונת הקלט, ולכל בית בשורה:

- נמצא ב- $O(1)$ את הערך שאליו ממופה בית מספר x בשורת הקלט.
- את הערך שמצאנו בשלב הקודם, נשים בבית מספר $N-x$ בשורת הפלט.

אפשרות שנייה:

א. נגדיר Bit-Field של 8 ביטים:

```
typedef struct
{
    unsigned char b1 :1;
    unsigned char b2 :1;
    ...
    unsigned char b8 :1;
} Byte;
```

ב. עבור כל שורה נבצע:

```
Byte* inputPtr = (Byte*)inputRow;
Byte* outputPtr = (Byte*)outputRow;
For (int i=0; i<N; i++)
{
    outputRow[N-1-i].b8 = inputRow[i].b1;
    outputRow[N-1-i].b7 = inputRow[i].b2;
    ...
    outputRow[N-1-i].b1 = inputRow[i].b8;
}
```

באפשרות השנייה מבצעים פי 8 יותר פעולות (השמה), אבל אין צורך בהכנת ה-Look-Up Table או במקום שהיא תופסת. מצד שני, בהנחה שהיא הרבה יותר קטנה מהתמונה עצמה, סיבוכיות הזמן והזיכרון שכרוכה בה היא שולית. בכל מקרה, בשתי האפשרויות, אם ההנחה הנ"ל נכונה, סיבוכיות הזמן היא $O(M*N)$ וסיבוכיות הזיכרון היא $O(M+N)$.

תשובה לשאלה 56 (באדיבות יוליה)

תחילה נמייין 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 קואורדינטת X זהה, נבצע מיון משני באופן הבא – נמקם תחילה את כל הנקודות שמתחילות קטע, ואחריהן את כל הנקודות שמסיימות קטע.

כעת נאתחל $\text{Count} = \text{MaxCount} = 0$, 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 המשני:
- 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 $\text{MaxCount} = 2$.
 - 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 1. במצב כזה, נרצה קודם להקטין את Count ב-1 ואחר כך להגדיל את Count ב-1 ולא להיפך, כדי שנקבל $\text{MaxCount} = 1$.
- הערה: בקואורדינטות ה-Y של הנקודות הנתונות אין שימוש כלל.

תשובה לשאלה 57

Constructor של מחלקה כלשהי לא יכול להיות וירטואלי, כי:

קריאה לפונקציה וירטואלית של אובייקט כלשהו מתבצעת בעזרת ה-V-Table של המחלקה שאליה שייך האובייקט. כל אובייקט מחזיק מצביע ל-V-Table המתאים, אבל המצביע הזה מאותחל רק בזמן ריצה, כאשר האובייקט נוצר. כלומר, המצביע הזה למעשה מאותחל רק ב-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 את הסוג המדויק של האובייקט שבעזרתו 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 שיוורשים 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 המצביע 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 $\text{ptr} = \text{new B}$ 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 58

על מנת למנוע פניה למערכת ההפעלה בעת 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 שנמשך 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 (במחשנית 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 59

שיטה ליישם ב-C++ את מנגנון הפונקציות הווירטואליות ללא שימוש במילה השמורה virtual:
מחלקה שמספקת ממשק של פונקציות וירטואליות, היא מחלקת בסיס למחלקות אחרות (שירשו 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 שירשו ממנה נאתחל את type.
מכיוון שה-Constructor הוא אף פעם לא 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.
נניח שמחלקה A היא מחלקת הבסיס, ושמחלקות B ו-C יורשו 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 את סוג האובייקט שלנו, ההמרה של ה-this למצביע מהסוג הזה והקריאה לפונקציה בעזרתו יהיו "חוקיות" (לא נקבל שגיאה בזמן ריצה).
דוגמא ליישום 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:

- 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, האופרטור מחזיר מצביע מתאים, ואם לא הוא מחזיר NULL. על מנת להשתמש באופרטור הזה, צריך לוודא שקיימת בקומפיילר אופציית RTTI, שמאפשרת לקבל אינפורמציה לגבי סוג האובייקט בזמן ריצה (Run-Time Type Information).
- באופן כללי, גם השימוש ב-RTTI וגם הדוגמא הנ"ל ליישום של פונקציה וירטואלית, נוגדים את הקונספט של OO. הרעיון בקונספט הזה הוא הכללה של אובייקטים דומים, ושימוש בהם בלי לדעת את הסוג המדויק של כל אחד (Polymorphism). כאן עשינו בדיוק את ההיפך – בדקנו את סוג האובייקט וביצענו פעולה מסוימת בהתאם.
- בנוסף, בכל פעם שנרצה לגזור (לרשת) מחלקה נוספת מ-A, נצטרך לשנות את הקוד של הפונקציות שבהן מימשנו את "מנגנון הווירטואליות". כלומר, נצטרך לקמפל וללנקג' גם את קוד המקור של A (בנוסף לקוד המקור של המחלקה החדשה).

תשובה לשאלה 60

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 ($\text{if } x < y$), נבצע סך הכל $\frac{1}{2}N$ השוואות. לאחר שקיבלנו $\frac{1}{2}N$ ערכי מינימום ו- $\frac{1}{2}N$ ערכי מקסימום:
 - נעבור על ערכי המינימום ונמצא את הערך המינימלי מביניהם – סך הכל $\frac{1}{2}N$ השוואות.
 - נעבור על ערכי המקסימום ונמצא את הערך המקסימלי מביניהם – סך הכל $\frac{1}{2}N$ השוואות.
- סך הכל, מספר ההשוואות שביצענו הוא $\frac{1}{2}N + \frac{1}{2}N + \frac{1}{2}N = 1\frac{1}{2}N$.

תשובה לשאלה 61

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

- 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.
 - בכל המשבצות בשורה הראשונה ובטור הראשון בטבלה נרשום 1, מכיוון שזהו אורך הצלע של הריבוע שכל אחת מהן מייצגת.
 - נעבור על הטבלה בצורה סדרתית החל מהשורה השנייה (משמאל לימין בכל שורה, לפי סדר השורות), ולכל משבצת $[i][j]$ נחשב את הערך המתאים לה באופן הבא:
 - נבדוק את הצבע של משבצת $[i][j]$ ושל שלוש המשבצות שלפניה $([i-1][j-1], [i-1][j], [i][j-1])$.
 - אם הצבעים של חלק מהמשבצות שונים זה מזה, אז משבצת $[i][j]$ מייצגת ריבוע שאורך הצלע שלו הוא 1.
 - אם הצבעים של כל המשבצות זהים זה לזה, אז משבצת $[i][j]$ מייצגת ריבוע שאורך הצלע שלו הוא $+ 1$ הערך המינימלי מבין הערכים שרשומים בשלוש המשבצות הנ"ל.
 - 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: $X = 1 + \text{Min}(1,2,2) = 2$.

תשובה לשאלה 62

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

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

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

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


תשובה לשאלה 63

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 שהתוכנה היא תוכנת Real-Time, נבצע את כל ההקצאות בזמן קומפילציה במקום בזמן ריצה (הקצאה סטאטית במקום הקצאה דינאמית), ע"מ להבטיח את מיקום כל האלמנטים בזיכרון.
- 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 64

```
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
```

את הפקודה CLR אפשר לממש בעזרת הפקודות DEC ו-JUMP. ביצוע הפקודה INC שקול לביצוע הפקודה DEC 15 פעמים, ולכן גם אותה לאפשר לממש בעזרת שתי הפקודות הנ"ל. אם קלט אחד הוא 0, אז הקלט השני יוכפל ב-16. מכיוון שהפלט נתון ברגיסטר של 4 ביטים, התוצאה עדיין תהיה 0. הקוד עובד גם עבור מספרים שליליים. ישנן רק 8 מכפלות אפשריות שבהן אין גלישה. לכן, מספיק לבדוק אותו עבור כל אחת מהמכפלות האלה: $-1*1, -2*1, -3*1, -4*1, -5*1, -6*1, -7*1, -8*1$.

תשובה לשאלה 67

פונקציה 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 לסוג האובייקט 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 68

ניעזר באבחנה שהמסלול 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 69

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.

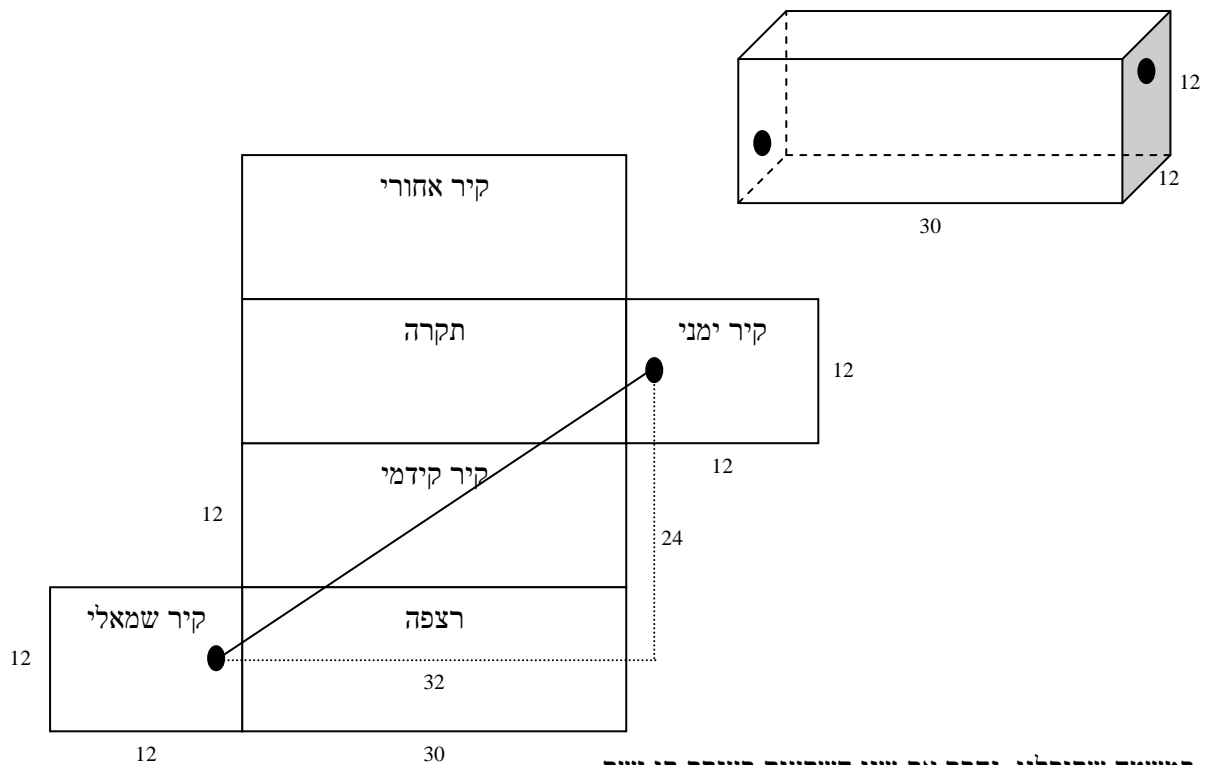
תשובה לשאלה 70

מספיק שאחד האנשים יוכל לחשב את ממוצע המשכורות. נבחר את האיש הראשון לצורך כך. מכיוון שהוא יודע מהי המשכורת שלו, מספיק לו לדעת מהו סכום המשכורות של השניים האחרים:

- האיש הראשון – בוחר מספר X כלשהו, מחבר אליו את המשכורת שלו ואומר את התוצאה לאיש השני.
- האיש השני – מחבר את המספר שקיבל מהאיש הראשון למשכורת שלו ואומר את התוצאה לאיש השלישי.
- האיש השלישי – מחבר את המספר שקיבל מהאיש השני למשכורת שלו ואומר את התוצאה לאיש הראשון.
- האיש הראשון – מחסר X מהמספר שקיבל מהאיש השלישי, מחלק את התוצאה ב-3 ומקבל את הממוצע.

תשובה לשאלה 71

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 פיתגורס: $\sqrt{32^2 + 24^2} = 40$.

תשובה לשאלה 72

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.

תשובה לשאלה 73

נניח שמספר הקלט הוא x , ומספר הביטים ב- x שערךם 1 הוא N . על מנת למצוא את N תוך שימוש ב- N איטרציות:
א. בכל איטרציה נאפס ב- x את הביט הימני ביותר שערךו 1, באופן הבא:

- $...10...0$ זה הייצוג הבינארי של x .
- $...01...1$ זה הייצוג הבינארי של $x-1$.
- נבצע פעולת and בין שני הערכים ונכניס את התוצאה ל- x : $x = x \& (x-1)$.
- ב. כאשר הערך של x יגיע לאפס, נדע שהערך של N הוא מספר האיטרציות שביצענו.