שאלה 1:

א)

מנגנון RTTI שזה קיצור ל Run-time type information הוא הוספת מפתח לכל ערך בזיכרון שיציין את הטיפוס של הערך, המפתח יכול להיות רפרנס ל “type descriptor” או יכול להכיל את כל המידע על טיפוס, בדרך כלל אחיד לכל הערכים בזיכרון מאותו טיפוס והמפתח הזה יפוענח בזמן ריצת התוכנית בשפות דינמיות וידוע לקומפיילר בשפות סטטיות.

ב)

כשמשתמשים במנגנון RTTI אנחנו נשלם על זה מחיר במקום (לכל טיפוס חייב להיות vtable) ומשלמים על זה זמן.  
בשפת C קיים עקרון “no hidden cost” שהוא שהמתכנת לא משלם על דברים שאינם חשופים לו, RTTI הוא נמצא בשפה ולא חשוף למתכנת ואין לו שליטה עליו אבל הוא משלם עליו, וזה סותר את העיקרון הנזכר למעלה.

ג)

ב garbage collection אנחנו לא רק צריכים למצוא את הערכים שהקצינו אלא יתר על זאת אנחנו חייבים הטיפוסים של הערכים הקשורים אליהם, בזיכרון יש לנו בתים וביטים אבל אנחנו לא יודעים מה הם מייצגים מנגנון RTTI יעזור לנו להתגבר על דבר זה ואז אנחנו נדע מה הערכים המקושרים לערכים שהקצינו.

ד)

למשל ב deep cloning, לכל ערך אנחנו ניגש לכל ערך ישיג ממנו, בלי ה RTTI אנחנו לא נדע את הטיפוס של ערכים אלה ואז לא נדע איך לגשת להם מתוך הבלוק המכיל בתים.

שאלה 2:

א)

|  |  |
| --- | --- |
| similarity | difference |
| הגודל של המערך לא משתנה אחרי האתחול | גודל המערך הסטטי ידוע בזמן קומפילציה לעומת המערך הדינמי שיוחלט על גודלו בזמן ריצה |
|  | המערך הסטטי מוקצה על ה data segment  המערך הדינמי מוקצה על ה heap segment |
|  |  |
|  |  |

ב)

|  |  |
| --- | --- |
| similarity | difference |
|  | האינדקסים ב stack based array הם יוחלטו בזמן יצור המערך אבל ב associative array הם יכולים להיות כל דבר ואז הסט שלהם לא חסום |
|  | הגודל של stack based array הוא לא ישתנה במהלך ריצת התוכנית אבל associative array הגודל שלה יגדל ויקטן לפי מספר האיברים בה (כי היא בדרך כלל ממושתת כטבלת ערבול. |
|  | כיוון ש associative array דורשת מבנה נתונים מתוחכם ולא פשוט היא לא יעילה כמו stack based array |
|  |  |

ג)

אלגוריתם stop and copy

1. תחצה את הheap לשני חצאים, נקצה את כל הערכים בחצי הראשון ונשים את החצי השני ב "המתנה".
2. כאשר החצי הראשון מתמלא נעתיק את כל האובייקטים ש RC שלהם גדול מ 0 שאנחנו יכולים לגשת אליהם לחצי השני
3. נחליף תפקידים בין שני החצאים.

נראה את היתרונות והחסרונות ל stop and copy עם mark and sweep

|  |  |
| --- | --- |
| cons | pros |
| בכל רגע נתון יהיה לנו חצי הזיכרון לעבוד איתו | זמן ריצת ה GC הוא לעומת ב mark and sweep |
|  | הפרגמנטציה קטנה |
|  | מימוש קל |

ראה את היתרונות והחסרונות ל stop and copy ביחס לניהול זיכרון ידני

|  |  |
| --- | --- |
| cons | pros |
| בכל רגע נתון יהיה לנו חצי הזיכרון לעבוד איתו | *חיים קלים למתכנת כי הוא לא ישחרר זיכרון בעצמו* |
| מאבדים יעילות של התוכנית בזמן ריצת GC הדבר הזה שיכולים לחסוך בו ע"י תכנות יותר טוב | לא יהיה heap corruption או memory leaks או dangling reference |
|  |  |