שאלה 1:

* בניית האובייקט Bar() תגרום להפעלת הבנאי Bar::Bar(), אך
* לפניו תתבצע קריאה לבנאי הריק של מחלקת האב כברירת מחדל (שהרי לא מוגדר אחרת) – Foo::Foo().
* בתוכו ישנה קריאה לFoo::baz().
* ולמרות זאת, היות וFoo::baz() **וירטואלית**, ואנו יצרנו במקור אובייקט מסוג Bar, יתבצע dynamic binding של הקריאה ל-baz() מתוך Foo,
* כלומר בפועל תבוצע קריאה לפונקציה Bar::baz()!
* ויודפס Bar::baz()!

שאלה 2:

אנו רוצים לקבל פלט כדוגמת:

**p1=(1,2)**

**p2=(5,2)**

וריצה מוצלחת של התוכנית.

אך בפועל, התוכנית קורסת, לאחר שהיא מדפיסה למסך:

**p1=(5,2)**

**p2=(5,2)**

כלומר, אנו רואים שרצינו לשנות רק בp2 רק את קואורדינטת ה-X, אך בפועל השתנתה גם קואורדינטת ה-X של p1. זהו רמז עבה לבעיית aliasing, ואכן, ניתן לראות כי בניית שני האובייקטים מתבצעת כך: בנייה שלp1 עם הבנאי שמקבל 2 שלמים, ובנייה של p2 עם השמה מיידית מ-p1. השמה זו, משום שמתבצעת באותה שורה שבה מוגדר המשתנה, גורמת לקריאה לבנאי של Point, בעל החתימה:

ונשים לב כי בבנאי זה – מבוצעת העתקה ישירה של המשתנה היחיד במחלקה, \_coord. אבל! משתנה זה הוא מערך של 2 שלמים, כפי שמוגדר בבנאים האחרים. כלומר, כאשר אנו פונים לp1 אשר מתקבל בבנאי זה, אנו פונים למערך \_coord שיצר הבנאי ביצירת p1, ומעתיקים את הכתובת שלו. על כן, נקבל כי \_coord מצביע בדיוק לאותה כתובת בזיכרון בשני האובייקטים. זה מסביר את העובדה שיש שינוי פרמטר אחד באובייקט מסוים, ובשני הוא משתנה גם.

בנוסף, זה מסביר את השגיאה שמונעת את סיום התוכנית בצורה תקינה! בגלל שp1,p2 משתנים לוקאליים, הם מושמדים ביציאה מהScope, במקרה זה – בסיום התוכנית והיציאה מ-main. נשים לב כי בצורה הגיונית, היות ואנו מקצים זיכרון בבנאים למערך \_coord, בהשמדה צריך לשחרר זיכרון זה – ואכן כך עושה ~Point. אך היות ובשני האובייקטים יש בדיוק אותו מערך \_coord ברמת הכתובת בזיכרון, ביצוע delete עליו בפעם הראשונה ישחרר אותו, ובפעם השנייה יגרום לחריגה – כי מנסים לשחרר כתובות שאינה מוקצית!