**קומפילציה – ממ״ן 16 – פרויקט גמר – README**

הפעלת הקומפיילר: הפרויקט נכתב בשפת Python, וניתן להפעיל אותו בעזרת גרסה 3.8 ומעלה של השפה. יש להתקין בנוסף את החבילה Sly (בגרסה 0.5 ומעלה) – בלבד.

דוגמת הרצה פשוטה:

$ python cpq.py input\_file.cpl

שלבי הקומפילציה:

1. מנתח לקסיקלי: המנתח הלקסיקלי פשוט למדי, ומהווה מימוש סטנדרטי של מנתח לקסיקלי כאשר משתמשים בספריית Sly לשם מטרה זו. יש לשים לב שמימוש ״התעלמות״ ב-Sly היא ע״י Token-ים שמתחילים במילה ignore, כנדרש.  
   המנתח הלקסיקלי מעניק ערכים לקסיקליים שימושיים וסבירים, שמספקים נקודת מבט מועילה על הToken עבור השלבים הבאים – בדיקת של מספר האם integer/float והמרה לכזה ברמת הטיפוס הפייתוני, המרה של אופרטורים בינאריים שונים ל-Enum גלובלי, חילוץ הטיפוס מ-static\_cast ושימוש בו כערך לקסיקלי (גם הוא Enum), והדפסת שגיאה שכוללת מס׳ שורה.
2. מנתח תחבירי: גם המנתח התחבירי מבוסס, באופן צמוד, על Sly לבנייתו. תפקידו של המנתח התחבירי, הוא אך ורק לבנות Abstract Syntax Tree מתוך התחביר, ולהתריע על שגיאות תחביר – Syntax Errors. הוא פשוט מאוד, ועל ה-AST מפורט בסעיף הבא.
3. סריקת Abstract Syntax Tree, וניתוח סמנטי (חלקי): על מנת לבצע את הסריקה על העץ, בניתי את מחלקת הבסיס AstNode, היא מהווה dataclass של פייתון, כמו Record Type בשפות עיליות שונות כמו C# או Java – כלומר, ישנו בינוי אוטומטי של Getters/Setters/CTOR/\_\_str\_\_. בנוסף לכך, הוספתי למחלקה יכולת חשובה מאוד – יכולת סריקה וביצוע Action-ים תוך כדי הסריקה (למעשה מימוש SDT). בעזרת דריסת הפונקציות before/after ניתן לבצע קוד בתחילת או סיום הסריקה (בסדר DFS עבור כל צומת, כאשר עוברים מה-Property הראשון ל-Property האחרון בעץ), ובעזרת הDecorator-ים @before\_visit/@after\_visit, ניתן לבצע קוד לפני או אחרי הסריקה של Property מסוים.  
   הסריקה כמובן רקורסיבית, ומבוצעת רק עבור Properties של המופע במידה והם בעצמם מהווים מופע של AstNode, או רשימה של מופעי AstNode – כמובן הכוונה ב״מופע״ – למופע של המחלקה עצמה או מחלקה יורשת כלשהי. תוך כדי הסריקה מבוצעת הפקת הקוד, כאשר הקוד נעזר בתוויות סמנטיות. המחלקה AstNode מספקת מעטפת ברורה וקלה למימוש ה-SDT.
4. המחלקה QuadCode שיש לה מופע בודד בכל קומפילציה מספקת שיטות לביצוע יצירת הקוד. בהקשר של הדפסת קוד – ישנן השיטות emit\* למיניהן, והן מתועדות בקוד עצמו. השיטות newtemp/newlabel יוצרות בהתאמה משתנה זמני (לפי טיפוס שלו) או תווית סמנטית כלשהי. השיטה emitlabel מדפיסה תווית סמנטית – כלומר ״מסמנת״ להיכן היא מצביעה מבחינת מספר של שורת קוד. ישנה מניעה של התנגשות בין שמות של תוויות ומשתנים זמניים למשתנים שהמשתמש מגדיר – על מנת להפוך את הקוד המופק לקריא ומובן לאחר הקומפילציה, אין בכלל ערבול של השמות – אלא להפך, אנו נעזרים בשמות מקוריים של משתנים, ככל שניתן. למעשה, לכל משתנה יש טיפוס מוגדר וחד משמעי – מהרגע שהוא מוכרז הוא נכנס לטבלת הסמלים, וכל משתנה זמני גם נכנס לטבלת הסמלים. באופן זה, תוך כדי הפקת הקוד, ישנן קריאות לשיטה הפנימית auto\_cast, שמוסיפה הוראות ITOR באופן אוטומטי במקרה הצורך, או מתריעה על שגיאה במצב של צורך להמרה מ-Float ל-Integer.
5. הקוד נכתב בכל emit כלשהו לתוך מבנה נתונים זמני – שמקל מעט על העיבוד ועל הכתיבה הסופית. המבנה הוא List[Tuple[QuadInstruction, Arg,Arg,Arg]]] כאשר Arg הינה מחרוזת או מספר – ארגומנט אפשרי להוראת Quad.
6. השיטה apply\_labels מחליפה תוויות סמנטיות במספרי השורות, כפי שהוגדרו ע״י emitlabel. כאמור, יש מניעת התנגשויות – ולכן מספיק מבט ב-Dictionary של התוויות כדי להיות בטוח האם יש להחליף מחרוזת כלשהי במבנה הנתונים, ביעד קפיצה מפורש כלשהו.
7. השיטה Write מדפיסה את הקוד לקובץ, מבצעת המרה מטיפוסים שיכולים להופיע במבנה הנתונים הזמני למחרוזות.
8. טיפול בשגיאות: הטיפול בשגיאות והתאוששות מהן נעשה בצורה הטובה ביותר, עד כדי הצורה שיכול לספק Sly בחלק של הLexer/Parser, ובהקשר של שגיאות סמנטיות – ישנו דגל רקורסיבי \_success שנבנה באופן הדרגתי ומהווה חלק מהמחלקה AstNode. במידה ונזרקת חריגה מסוג SemanticError היא נתפסת והדגל הופך לשלילי, וכך גם אם מודפסת שגיאה כלשהי! ברור כי אם נזרקת חריגה ה-Parsing עוצר, ולכן חריגה כזאת נזרקת רק מתוך המחלקה QuadCode – רק שם יש צורך, בהקשר של טיפוסי משתנים והמרתם.
9. מבני הנתונים: כאמור, הקוד ממומש ע״י רשימת פייתון, היא המבנה הפשוט והיעיל ביותר לאחסון הקוד כולו – מאפשרת גישה מהירה יחסית לאינדקסים מסוימים, ודינאמיות מלאה. טבלאות הסמלים והתוויות הן Dictionary – כלומר Hash Table – מבנה נתונים עם זמן גישה קבוע ואופטימלי, כמובן בגודל דינאמי. שימוש מרובה ב-Enum-ים להפשטה של הקוד, הבהרה של הקבועים ומניעת טעויות נעשה באופן נרחב.