בס"ד

תיאור הפרויקט

הפרויקט נכתב בשפת C# תוך שימוש בכלי ANTLR ב Visual Studio 2019.

מבנה הקוד

הכלי ANTLR מקבל כקלט cpl.g4 (מכיל את הדקדוק של שפת cpl) ומייצר את הקבצים הבאים (מזכירה כאן את הקבצים העיקריים החשובים):

- CPLLexer.cs
- CPLParser.cs
- CPLVisitor.cs

הקבצים שנוצרים על ידי ANTLR נמצאים בתיקיה Generated files.

מחלקות מרכזיות

- Program - נקודת הכניסה של הקומפיילר. מקבל כארגומנט את הקובץ לקימפול. מבצע ולידציה לפני שליחה לתהליך הקימפול.
- CplCompiler - יוצר parser לקוד. ה parser מכיל בתוכו גם את ה lexer. לאחר הרצת ה parser מתקבל AST עליו עוברים ב CplVisitor.
- CplVisitor - עובר על עץ הקוד (AST) שנוצר על ידי ה parser. מכיל מתודות visit. לכל כלל בדקדוק יש מתודת visit בהתאמה. לדוג' עבור הכלל

  program -> declarations stmt_block

  מוגדרת מתודת visit:

  public override object VisitProgram(ProgramContext context)

  במתודות visit מתבצע התרגום משפת cpl ל quad.
  מחלקת CplVisitor יורשת ממחלקת CplBaseVisitor שהוא קובץ שנוצר על ידי ANTLR (על סמך הדקדוק שסופק) ובו מוגדרות כל מתודות ה visit כ virtual.
- CplVisitorUtils - חלק ממחלקת CplVisitor. מוגדר בקובץ נפרד (מחלקת CplVisitor מוגדרת כ partial class כדי שיהיה ניתן לפצל את המחלקה לכמה קבצים עקב גודלה). מכיל מתודות עזר לצורך תרגום הקוד לשפת quad.
- FloatType/IntType - ממשמ את הממשק IType. מכיל את קוד הפקודות בשפת quad לפי סוג המשתנה. לדוג' פקודת הדפסה בשפת quad תמומש במחלקה IntType כ IPRT וב FloatType כ RPRT.
- SyntaxErrorListener - מדפיס ל console במקרה של שגיאה תחבירית בשלב ה parser.
- SemanticErrorHandler - מדפיס ל console במקרה של שגיאה סמנטית (כגון הצהרה כפולה על אותו משתנה) בשלב המעבר על ה AST.

מימוש

- **שמירת הקוד המתורגם** מכיוון שלא תמיד יש את כל המידע הדרוש לכתיבת פקודה מסויימת (כגון פקודת קפיצה) ולאחר מכן יש צורך לעדכן את הפקודה עם המידע החסר, לכן בחרתי לשמור את הקוד המתורגם כרשימה של מחרוזות. בסוף התרגום רושמים כל מחרוזת כשורה נפרדת בקובץ ה output. מכיוון שבחרתי לעבוד עם רשימת מחרוזות, אין צורך לנהל את מספר השורות - ניתן לחישוב פשוט על ידי האינדקס של מחרוזת הפקודה ברשימה.
- **שמירת המשתנים** (symbol table) נעשית במבנה מסוג מילון (dictionary). המפתח (key) הוא שם המשתנה והערך (value) הוא הסוג של המשתנה. בחרתי להשתמש במבנה נתונים זה כיוון שהגישה למשתנה קורית ב $O(1)$.
- **הצהרות ומזהים** עבור כל הצהרה או שימוש במזהה, מתבצעת בדיקה האם המזהה קיים בטבלת הסמלים. במידה והמשתנה קיים כבר, מוחזרת הודעת שגיאה. במידה והמשתנה לא קיים, מוסיפים את המשתנה לטבלת הסמלים.
- **שמירת טיפוס הנתונים** מכיוון שלכל פקודה בשפת quad יש שני מימושים אפשריים, סוג המשתנה נשמר גם כן בטבלת הסמלים ולפני כל תרגום מתבצעת שליפת סוג המשתנה מטבלת הסמלים (לפי שם המשתנה) כדי לדעת איזו פקודה להדפיס..
- **משתנים זמניים** לעיתים יש צורך בהוספת משתנה זמני. משתנה כזה מסומן כ t_x כאשר $x \geq 0$. דוגמה למשתנה זמני: t_0. יצירת משתנה זמני חדש מתבצעת על ידי המתודה GetTempVar.
- **טיפול ב expressions** שמירת ה expressions נעשית במבנה נתונים מסוג מחסנית. כל אלמנט במחסנית הוא מסוג keyValuePair כאשר המפתח (key) הוא שם המשתנה והערך (value) הוא הסוג של המשתנה. בחרתי להשתמש במבנה נתונים זה כיוון שסדר הטיפול במשתנים צריך להיות LIFO. כמו"כ, פעולות push ו pop קורות ב $O(1)$.

הכנסת משתנה למחסנית קורית כשמגיעים למתודת VisitFactor. ההחלטה על סוג המשתנה נקבעת בצורה הבאה:

★ במקרה של ID - לפי הסוג שמופיע בטבלת הסמלים.

★ במקרה של NUM - במידה והמספר כולל נקודה עשרונית - הסוג יהיה float, אחרת int.

הוצאת משתנה מהמחסנית קורית במתודות VisitTerm ו VisitExpression בעת תרגום פקודות חיבור, חיסור, כפל, חילוק.
כיוון שכל אחת מהפעולות הנ"ל מצריכה משתנה זמני, גם המשתנה הזמני מוכנס למחסנית לאחר תרגום הפקודה.

- **השמה** כאשר מנסים לבצע השמה מ float ל int, מוחזרת הודעת שגיאה. במידה ויש השמה מ int ל float, מתבצעת המרה מרומזת ומוסיפים פקודת ITOR לפני ביצוע ההשמה.
- **ביטויים בוליאנים** המתודות שמטפלות בביטויים בוליאנים הן VisitBoolTerm, VisitBoolExpr ו VisitBoolFactor.

★ במתודה VisitBoolFactor מתבצעת קריאה למתודה הבסיס VisitBoolFactor שממנה קוראים בסופו של דבר למתודה VisitTerm. כלומר, רק לאחר הקריאה ל base.VisitBoolFactor מתבצעת הכנסת ה expressions למחסנית. לאחר שה expressions הוכנסו למחסנית, ניתן לטפל בהם בהתאם לאופרטור היחס (relational operator) המתאים. הטיפול נעשה במתודה HandleRELOP. לאחר כתיבת הפקודה המתאימה דוחפים את המשתנה הזמני למחסנית ה expressions.

★ במתודה VisitBoolTerm מטפלים באופרטור AND. שולפים את שני המשתנים שנוספו אחרונים למחסנית ה expression וכותבים שתי פקודות:

- ○ פקודת חיבור של שני המשתנים.
- ○ פקודה שבודקת האם תוצאת החיבור גדולה מ 1 (כלומר שבדיוק שני התנאים התקיימו).

★ במתודה VisitBoolExpr מטפלים באופרטור OR. שולפים את שני המשתנים שנוספו אחרונים למחסנית ה expression וכותבים שתי פקודות:
  - פקודת חיבור של שני המשתנים.
  - פקודה שבודקת האם תוצאת החיבור גדולה מ 0 (כלומר שלפחות אחד משני התנאים התקיים).

- **חישוב יעד קפיצה במשפט if** בתרגום משפט if יש לחשב שני יעדי קפיצה:

★ פקודת JMPZ - למקרה שהתנאי לא מתקיים. מספר השורה אליה יש לקפוץ ידוע רק לאחר כתיבת בלוק האמת ולכן עדכון פקודת JMPZ יבוצע אח"כ.

★ פקודת JUMP - נכתבת בסוף בלוק האמת לשורה הבאה לאחר משפט ה if (כדי לדלג לקוד שאחרי ה else). מספר השורה אליה יש לקפוץ ידוע רק לאחר כתיבת בלוק ה else ולכן עדכון פקודת JUMP יבוצע אח"כ.

מכיוון שבשלב כתיבת פקודת הקפיצה מספר השורה אינו ידוע, נשמור מצביע לאינדקס של פקודת הקפיצה ברשימת הפקודות על מנת לעדכן אותה כשמספר השורה יהיה ידוע.

- **חישוב יעד קפיצה במשפט while** בדומה למשפט if, בתרגום משפט while יש לחשב שני יעדי קפיצה:

★ פקודת JMPZ - במקרה שתנאי הלולאה לא מתקיים - מספר השורה אליה יש לקפוץ ידוע רק לאחר כתיבת גוף הלולאה ולכן עדכון פקודת JMPZ יבוצע בסוף כתיבת בלוק ה while.

★ פקודת JUMP (לתנאי הלולאה) - נכתבת בסוף כתיבת בלוק גוף הלולאה.

- **טיפול במשפט break** משפט break יכול להופיע בלולאת while או במשפט switch. במתודה VisitBreak_stmt מוסיפים למחסנית אלמנט מסוג Tuple השומר שני שדות:

★ שדה המונה באיזה משפט while\switch כתוב משפט ה break. בכל כניסה למשפט while\switch מעלים את המונה ב 1 ובכל יציאה ממשפט while\switch מורידים את המונה ב 1. לכל משפט יש מונה נפרד. דוגמה: אם התוכנית מכילה משפט while בתוך משפט while אחר ובמשפט ה while הפנימי מופיע משפט break, הערך שנשמור במחסנית יהיה 2.

★ מצביע לאינדקס של פקודת הקפיצה ברשימת הפקודות על מנת לעדכן אותה כשמספר השורה יהיה ידוע. עדכון מספר השורה נעשה על ידי המתודות VisitWhile_stmt ו VisitSwitch_stmt ולכן מוגדרות שתי מחסניות לשמירת ה indexes - אחת למשפט break בתוך לולאת while והשניה עבור משפט break בתוך switch. ההחלטה לאיזו מחסנית להוסיף את האינדקס נעשית על ידי זיהוי ה context בו כתוב ה break.

- **טיפול במשפט switch** במשפט switch יש לתרגם משפט case לתנאי השוואה בין המספר לתוצאת ה expression ולהוסיף משפט JMPZ ל case הבא. תרגום משפט break יהיה פקודת JUMP למשפט הבא לאחר פקודת switch.