ביה"ס תיכון ע"ש הרצוג כפ"ס

מגמת הנדסת תוכנה וסייבר

פרויקט גמר

פירוט ועיצוב תוכנת פרויקט בנושא:

"קומפיילר ושפת תכנות Gravity"

פיתוח מהדר ויצירת שפת תכנות - פרסור, אנליזה וגנרציה

מגיש: אופיר הופמן

ת.ז: 217395094

בהנחיית המורה:

אופיר שביט

רקע

הפרויקט עוסק בפיתוח קומפיילר ומתפצל לשלושה נושאים – פיתוח הקומפיילר, עיצוב שפת התכנות ופיתוח סביבת עבודה ב-Web.

פיתוח הקומפיילר: קוד הקומפיילר יפוצל לפי שלבי הקומפילציה. כלומר, תהליך בניית הקומפיילר יתחלק לפיתוח כל הרכיבים בתהליך הקמפול בנפרד, כך שיעבדו בצורה אינדיבידואלית. כל כלי שאפתח משרת מטרה אחרת בשלב הקמפול וביחד הכלים יעבדו בצורה כזו שתוכנה אחת מקבלת קלט מסויים, מעבדת אותו ומעבירה את הפלט שלה לתוכנה הבאה. כך עד שלבסוף ייצא קלט של קובץ הרצה בינארי (או משהו אחר בהתאם לבקשת המשתמש).

עיצוב שפת התכנות: היות ובפרויקט זה אני יותר שפת תכנות חדשה לחלוטין, יש להחליט על נראותה – סינטקס, סמנטיקה, פרגמטיקה וכו'. ארצה שבפרויקט זה השפה תהיה שונה קמעה משפות קיימות, אך עליה להיות גם נוחה לשימוש והבנה בכך שתישען על הפרגמטיקה של שפות high-level מוכרות.

סביבת עבודה ב-Web: אתר עם ממשק משתמש יפה ונוח שיאפשר לכתוב ולערוך קוד בשפה, ולסיום יוכל המשתמש לקמפל אותו "מרחוק" ולקבל בחזרה את הפלט אל האתר. מאחורי הקלעים המערכת הזו תעבוד עם שרת מרוחק אשר יהיה אחראי לקבל בקשות http/WebSocket ולהגיב להן בהתאם.

סביבה

- **שפות:** לכל שלב בתהליך ההידור ייתכן שימוש בשפה אחרת, באופן כללי אשתמש ב-C/C++ לקוד ה-main של רוב הכלים. שפות אלו מאפשרות ביצועים מהירים וגישה יותר נוחה וברורה ל"מאחורי הקלעים" של התוכנה, אשר לטעמי חשובה בכתיבת פרויקט זה.
- לסביבת העבודה ב-Web אשתמש ב-HTML, CSS ו-javascript.
- **סביבות עבודה**: VS Code, notepad++
- **מערכות הפעלה:** Windows (תאימות מלאה למערכת מבוססת דפדפן).

טכנולוגיות

- **Flex:** משמש לניתוח לקסיקלי ויצירת טוקנייזר מותאם אישית עבור השפה.
- **ANTLR:** משמש ליצירת פרסר עבור השפה, מאפשר תמיכה בסינטקס מותאם אישית.
- **Bison:** משמש לניתוח סינטקטי ולבניית AST (Abstract Syntax Tree) מדויק.
- **פרוטוקולים:WebSocket/HTTP** לתקשורת בזמן אמת בין הלקוח לשרת לצורך סביבת העבודה ב-Web.

ארכיטקטורת מערכת

הארכיטקטורה והעיצוב הטכני

ארכיטקטורת מערכת

מערכת הקומפיילר כוללת שלושה חלקים עיקריים, המחולקים בהתאם לתפקודיהם המרכזיים:

1. **ממשק משתמש (Frontend)**
   - **אחראי על איסוף קלט מהמשתמש, הצגת תוצאות הידור ושגיאות, ותפעול כלים אינטראקטיביים כמו דיבאגינג.**

2. שרת (Backend)
   - מבצע פעולות תקשורת לקבלת בקשות והחזרת תגובות ללקוחות.
   - מיושם באמצעות python ופרוטוקלי תקשורת לדוגמת HTTP/WebSocket
3. מנוע קומפילציה
   - כולל את כלים כמו Flex ו Bison-לניתוח לקסיקלי וסינטקטי וגנרציה.

מודולים

מודולים בFrontend-

1. Editor
   - מבוסס על.Monaco Editor
   - מציג תחביר מודגש, הצעות אוטומטיות, ותצוגת שגיאות.
2. Debugger
   - מספק כלים להרצת קוד צעד-אחר-צעד עם הצגת מצב משתנים.

מודולים בBackend-

1. Parser Module
   - מנתח את התחביר של הקוד.
   - מייצר עץ תחבירי (AST) באמצעות Flex ו.Bison-
2. Semantic Analyzer
   - בודק את נכונות הקשרים הלוגיים והסמנטיים בקוד.
3. Code Generator
   - ממיר את עץ התחביר לקוד מכונה או בייט-קוד.
4. Project Manager
   - מנהל יצירה, טעינה ושמירה של פרויקטים.
5. Logger
   - רושם הודעות שגיאה ותהליך.

מבנה הפעולה הראשית

תהליך הידור קוד:

1. המשתמש מזין קוד בעורך.

2. הקוד נשלח לשרת דרך.WebSocket (במידה ומדובר בשימוש בסביבת העבודה ב-Web)
3. בשרת, מתבצע ניתוח לקסיקלי וסינטקטי ליצירת AST
4. ניתוח סמנטי מבוצע על AST לאימות תקינות לוגית.
5. הקוד מתורגם לבייט-קוד או לקוד מכונה.
6. הפלט מוחזר ל.Frontend-

תרשים זרימה:

- Input (User Code) → Frontend (Editor) → Backend (Parser → Semantic Analyzer → Code Generator) → Output (Compiled Bytecode).

מחלקות(UML)

1. Compiler
   - פעולות :
     - parse(input: string): AST
     - analyze(ast: AST): SemanticTree
     - compile(tree: SemanticTree): Bytecode
   - טענות :
     - parse: קלט הוא מחרוזת קוד, מחזיר עץ תחבירי חוקי.
     - compile: מחזיר Bytecode תקני בלבד.
2. ProjectManager
   - פעולות :
     - createProject(name: string): boolean
     - loadProject(id: number): Project
     - saveProject(project: Project): boolean
   - טענות :
     - loadProject: מחזיר פרויקט קיים או שגיאה.
3. Logger
   - פעולות :
     - logError(message: string): void

- getLogs(): Log[]

קבצים וטבלאות

קבצים

- Source Code Files: קבצי src. המייצגים קוד מקור.
- Bytecode Files: קבצי bc. המכילים את הפלט המהודר.

טבלאות SQLite

- Logs
  - שדות :
    - Timestamp: זמן יצירת הלוג.
    - Message: תוכן ההודעה.

פעולות עזר

1. tokenizeLine(line: string): Token[]
   - מבצע טוקניזציה לשורת קוד אחת.

הרחבות עתידיות

- הוספת תמיכה בסימולטור להרצת בייט-קוד.
- שילוב בדיקות יחידה אוטומטיות בקומפיילר.

תקשורת

פרוטוקול תקשורת - בין הלקוח לשרת

1. מטרה

מטרת פרוטוקול התקשורת בין הלקוח (Frontend) לשרת (Backend) היא לאפשר העברת בקשות להרצת קוד, שליחת נתונים, קבלת פלטים ותשובות מהשרת בצורה ברורה, עקבית ובטוחה. הפרוטוקול יאפשר למערכת לתפקד בצורה אינטראקטיבית, תוך שמירה על עקרונות יעילות וביצועים. התקשורת תתבצע באמצעות פרוטוקול HTTP או WebSocket, תלוי בצורך.

2. הודעות העוברות בתקשורת

ההודעות בין הלקוח לשרת תתבצע באמצעות שליחה וקבלה של בקשות ותגובות. יש להגדיר את סוגי ההודעות, כיוון התקשורת, והאם התקשורת סינכרונית או אסינכרונית.

סוגי הודעות:

- בקשה להרצה (Execute Request): בקשה של הלקוח לשרת להריץ קוד בשפת התכנות החדשה.

- תשובה/פלט :(Response) תשובה מהשרת, הכוללת את תוצאות הקוד או שגיאות שהתרחשו.
- הודעת סטטוס :(Status Message) הודעה המועברת בין הלקוח לשרת במהלך תהליך, לדוג' "הקוד ממתין לעיבוד" או "הקוד הושלם בהצלחה."
- בקשה לשמירה :(Save Request) בקשה מהלקוח לשמור קוד שנכתב.
- בקשה לשליפת היסטוריה :(History Request) בקשה לשליפת קוד שמור או מידע קודם.

כיוון ההודעות:

- כיוון מהלקוח לשרת :כל הבקשות (Execute Request, Save Request, History Request).
- כיוון מהשרת ללקוח :כל התשובות.(Response, Status Message)

סינכרוני/אסינכרוני:

- סינכרוני :בקשות להרצה ידרשו מענה מידי מהשרת (למשל, הפעלת קוד ידרוש תשובה מידית).
- אסינכרוני :בקשות לשמירה או שליפת היסטוריה יכולות להיות אסינכרוניות, במובן שמצופה לעדכן את הממשק בהצלחה או בשגיאה לאחר עיבוד השרת.

.3מבנה ההודעה וכל שדה בהודעה

לכל סוג הודעה יש מבנה ברור המורכב משדות שונים. להלן הגדרת המבנה עבור סוגי ההודעות המרכזיות.

הודעת בקשה להרצה:(Execute Request)

```
{
"request_id": "123456",
"type": "execute",
"code": "print('Hello, world!')",
"language": "myLang",
"timestamp": "2024-12-25T15:30:00Z"
}
```

- request_id:מזהה ייחודי לכל בקשה) מספר או.(GUID
- type:סוג הבקשה) במקרה זה.("execute" ,
- code:הקוד של המשתמש שצריך להיעבד.

- language: שפת התכנות של הקוד.
- timestamp: חותמת זמן של שליחת הבקשה.

הודעת תשובה:(Response)

```
{
"request_id": "123456",
"status": "success",
"output": "Hello, world!\n",
"errors": [],
"execution_time": "0.15s"
}
```

- request_id: מזהה הבקשה המקורי.
- status: סטטוס ההרצה) יכול להיות "success" או."error")
- output: פלט הקוד אם ההרצה הצליחה.
- errors: רשימת שגיאות אם היו (אם יש).
- execution_time: זמן ההרצה של הקוד.

4.הגדרה של שדות וערכים

לכל שדה בהודעה יש להגדיר את הגודל, צורת ייצוג המידע, והמפריד אם ישנו.

שדות בהודעת בקשה להרצה:

- request_id: מחרוזת באורך 36 תווים (UUID) המייצגת מזהה ייחודי.
- type: מחרוזת באורך משתנה) ערך קבוע כמו."execute", "save")
- code: מחרוזת באורך משתנה, המייצגת את הקוד ששלח המשתמש.
- language: מחרוזת המציינת את השפה) למשל."myLang" ,
- timestamp: תאריך ושעה בפורמט) ISO 8601 לדוג' "2024-12-25T15:30:00Z").

שדות בהודעת תשובה:

- request_id: מחרוזת באורך 36 תווים.(UUID)
- status: מחרוזת "success") או."error")
- output: מחרוזת שמכילה את הפלט או הודעת שגיאה.

- **errors: מערך של אובייקטים או מחרוזות, שיכולים לכלול שגיאות אם היו.**
- **execution_time: מחרוזת בפורמט זמן, לדוג' "0.15"s.**

5. רשימת שגיאות אפשריות בתקשורת

במהלך התקשורת עשויות להתעורר שגיאות שצריך לתעד ולספק לקוד מענה ברור. להלן קוד שגיאה נפוץ והסבר עבור כל קוד.

- **400 (Bad Request): בקשה לא תקינה. לדוג' אם הפלט לא נמצא או שהשדה code חסר .**
  - **הסבר: "הבקשה לא מכילה את כל השדות הדרושים".**
- **404 (Not Found): לא נמצא נתון, לדוג' קוד לא קיים או היסטוריה חסרה .**
  - **הסבר: "לא ניתן למצוא את הקוד המבוקש".**
- **500 (Internal Server Error): שגיאה פנימית בשרת, לדוג' כשל בתהליך הקומפילציה .**
  - **הסבר: "התרחשה שגיאה בשרת במהלך עיבוד הבקשה".**
- **408 (Request Timeout): בקשה לא הושלמה בזמן .**
  - **הסבר: "הבקשה לא הושלמה בזמן הקצוב. נא לנסות שוב מאוחר יותר".**