Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ПЕРЕКЛАД ДЖЕРЕЛЬНОГО КОДУ МІЖ МОВАМИ JAVA ТА PYTHON

Текстова частина до курсової роботи

за спеціальністю „Комп'ютерні науки"

Керівник курсової роботи

к.т.н., доц. _____

(прізвище та ініціали)

_____

(підпис)

"____" _____ 2020 р.

Виконав студент _____

_____

(прізвище та ініціали)

"____" _____ 2020 р.

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мультимедійних систем,

проф., д.ф.-м.н.

_____В. В. Бублик

(підпис)

„____” _____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту_____ факультету _____ _____курсу

ТЕМА _____

Вихідні дані:

-

-

Зміст ТЧ до курсової роботи:

Вступ

1 Огляд

2 Теорія

3 Реалізація

Висновки

Список літератури

Дата видачі „___” _____ 2020 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Тема: _____

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	02.02.2020	
2.	Огляд технічної літератури за темою роботи.	04.02.2020	
3.	Розробка внутрішніх систем	25.02.2020	
4.	Розробка MVP	30.03.2020	
5.	Розробка програмного інтерфейсу	15.04.2020	
6.	Розробка прикладної гри	10.04.2020	
7.	Написання пояснювальної роботи.	22.04.2020	
8.	Створення слайдів для доповіді та написання доповіді.	04.05.2020	
9.	Аналіз отриманих результатів з керівником, написання доповіді та попередній захист магістерської роботи.	07.05.2020	
10.	Корегування роботи за результатами попереднього захисту.	09.05.2020	
11.	Остаточне оформлення пояснювальної роботи та слайдів.	11.05.2019	
12.	Захист проекту	23.05.2019	

Студент _____

Керівник _____

“_____” _____

План

Вступ
1. Теоретичні особливості розробки транспілятора
    1. Транспіляція
    2. Аналіз мови Python
    3. Аналіз мови Java
    4. Відмінності між Python та Java
2. Реалізація транспілятора
    1. Загальний огляд транспілятора
    2. Побудова токенізатора
    3. Парсинг
    4. Створення проміжного представлення програми
    5. Побудова джерельного коду
    6. Заплановані особливості та функції
    7. Обмеження
3. Опис використаних технологій та їх ролі

Висновки

Джерела

Вступ

При проектуванні програмних проектів буває складно передбачити об’єм проекту та його вимоги, тож вибір мови та інструментів може бути не очевидним. Під час розробки таких проектів, розробники можуть зіштовхнутися з рядом проблем, пов’язаних з неоптимальним вибором мови, у результаті чого може виникнути потреба у переписуванні усього проекту на іншій мові програмування. Цей процес може займати значну кількість часу та коштів.

Транспілятори вирішують цю проблему автоматизуючи переклад вихідного коду однієї мови на іншу. Робота транспілятора у такому випадку є значно ефективнішою за ручний переклад. Є також і інші причини для міграції програмного коду з однієї мови на іншу:

- необхідність використання нової мови
- необхідність підтримки старшої версії використовуваної мови
- використання інструментів з екосистеми іншої мови
- збільшення продуктивності роботи програми

У цій роботі було досліджено особливості мов програмування Python та Java, та було створено програму транспілятор між цими двома мовами.

1. Теоретичні особливості розробки транспілятора

1.1 Транспіляція

Транспіляція (або “Source-to-Source” компіляція) — це процес перетворення вихідного коду програми, написаної однією мовою, на еквівалентний вихідний код на іншій мові програмування. У транспіляції мова, на якій написаний програмний код називається вхідною, а мова, на яку виконуєтсья переклад — вихідною. Подібно до компілятора, робота транспілятора полягає у синтаксичному та семантичному аналізі вхідного коду, побудові представлення структури програми та генерації вихідного коду. Але різниця між компіляторами та транспіляторами полягає у тому що компілятори виконують переклад з мови вищого рівня абстракції на мову нижчого рівня (наприклад переклад з C у машинний код, або з Java у код для віртуальної машини Java), тоді як транспілятор перекладає між мовами подібного рівня абстракції, але не обов’язково одинакового (наприклад переклад з Python на Javascript).

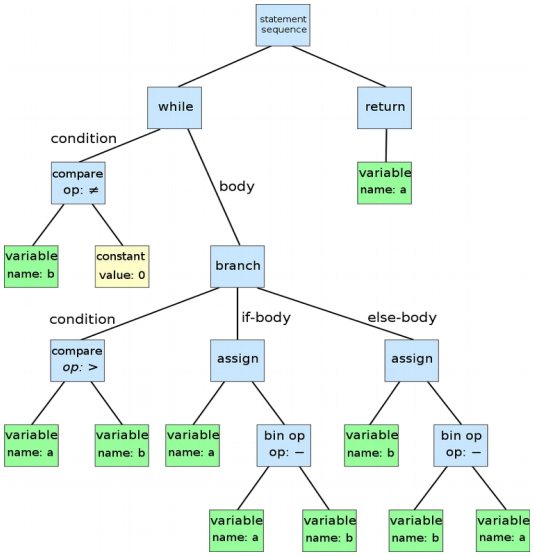
Процес транспіляції складається з наступних етапів:

1. Лексичний розбір

- Побудова абстрактного синтаксичного дерева програми для вхідної мови
- Побудова абстрактного синтаксичного дерева програми для проміжного представлення
- Побудова вихідного коду на вихідній мові з абстрактного синтаксичного дерева проміжного представлення

Лексичний розбір (токенизація) — це процес лексичного аналізу, у якому виконується перетворення послідовності символів у послідовність лексичних елементів (токенів). Лексичний розбір може бути першим кроком транспіляції, або відбуватися одночасно з наступним кроком: семантичним аналізом.

Семантичний аналіз (парсинг) — це процес аналізу вхідної послідовності лексичних елементів та побудова граматичної структури згідно з формальною



граматикою мови. У парсерах мов програмування ця синтаксична структура є абстрактним синтаксичним деревом[1]. Абстрактне синтаксичне дерево (АСД) — це орієнтоване дерево, побудоване за граматичними правилами мови програмування, у якому кожна вершина відповідає граматичній одиниці написаної програми. Ця структура даних широко використовуються у парсерах для представлення структури програми.

Перетворення синтаксичного дерева, побудованого за граматичними правилами вхідної мови, на граматичне дерево проміжного представлення виконується з метою спрощення генерації вихідного коду. Граматичні правила для побудови АСД проміжного представлення визначені таким чином, щоб було можливо побудувати вихідний код програми без проведення додаткового семантичного аналізу.

```
C:\Users\Igor4\Desktop\Payload3>lua tpile.lua simple.py
Tokenizing...
Parsing...
Compiling to internal...
Done
```

simple.py

```
print("Hello world!")
```

Main.java

```
public class Main
{
    public static String print(String s)
    {
        System.out.println(s);
    }
    public static void main(String[] args)
    {
        print("Hello world!");
    }
}
```

Побудова вихідного коду на вихідній мові є останнім кроком роботи транспілятора. Вона відбувається за рахунок проходу АСД проміжного представлення в глибину та генерації послідовності символів для вихідного коду. Вершини АСД, які, відповідно до граматичних правил, не містять корисної інформації для побудови результуючої послідовності ігноруються. Результатом роботи транспілятора є послідовність символів, яка є вихідним кодом програми, семантично одинаковим з вхідним кодом.

Для створення транспілятора потрібно провести аналіз специфікації та визначити граматичні правила кожної мови програмування, яка буде підтримуватись системою. За визначеними правилами для усіх мов потрібно побудувати токенізатори, парсери, генератори АСД проміжного представлення та вихідного коду. У даній роботі був проведений аналіз двох мов: Python та Java. За результатами цього аналізу був розроблений транспілятор, який здатний виконувати переклад коду, написаного на мові Python на відповідний код на мові Java.

1.2 Аналіз мови Python

Python – це високорівнева мова програмування загального призначення. Вона підтримує процедурну, об'єктно-орієнтовану, аспектно-орієнтовану та функціональну парадигми програмування. Python є динамічно типізованою мовою та використовує збирач сміття для керування ресурсами пам'яті. Філософія та принципи дизайну мови, описані у документі "Zen of Python"[2], є такими:

- Красиве краще, ніж потворне.
- Явне краще, ніж неявне.
- Просте краще ніж складене.
- Складене краще ніж складне.
- Простота читання рахується.

Будучи створеним за цими принципами, Python прагне до більш простого синтаксису та граматики, надаючи розробникам вибір для власної методології програмування. Вона також є простою для читання мовою: відсутність описів типів, використання роздільника для виразів не є обов'язковим, та розділ

```
1  a = 6
2
3  def fibonacci(n):
4      if n > 1:
5          return fibonacci(n-1)+fibonacci(n-2)
6      else:
7          return n
8
9  print fibonacci(a) # prints "8"
```

програмних блоків визначається за допомогою відступу.

Python використовує неявну типізацію. Змінним не надаються обмеження їх типів під час інтерпретації. Натомість, під час виконання, операції над об'єктом, непередбачені його типом, можуть привести до вийнятку або

аварійної зупинки програми. Python також є строго типізованою мовою та не надає можливості проводити нечітко визначені та невизначені операції. Користувачі здатні визначити власні типи даних використовуючи класи. Структура програм, написаних на Python має наступний вигляд:

- На найвищому рівні ієрархії є головний блок програми. У Python не існує поняття ‘функції точки входу’; виконання програми відбувається з найвищого блоку, яким є головний блок програми. Також у цьому блоці переважно виконується імпорт додаткових бібліотек, визначення класів та функцій.
- Кожний блок програми (враховуючи головний блок) складається з послідовності речень. Реченням може бути звичайне присвоєння значення змінній, виклик функції, повернення з функції (return), або будь-яке розгалуження (if, for, while…)
- Вирази у мові можуть складатися з констант, змінних або викликів функцій, та можуть мати довільний розмір, якщо між кожним елементом виразу є оператор. Вирази використовуються у присвоєннях, аргументах викликів функцій та при поверненнях значень з функцій.
- Змінні у Python використовуються у присвоєннях, виразах та при виклику функцій. Присвоєння значення змінній та оголошення змінної зі значенням синтаксично не відрізняються. Будучи динамічно типізованою мовою, змінна може змінити свій тип якщо їй присвоїти вираз іншого типу.
- Функції мають назву, перелік назв аргументів та блок функції. Визначення функції є реченням, тож її можна визнчаити у будь-якому блоці. Функції у Python є об’єктами першого класу, тож їх також можна присвоїти до змінних.

- Класи мають назву, перелік назв базових класів та блок класу, у якому визначені його методи. Подібно до функцій, класи можна визнчаити у будь-якому блоці та присвоїти до змінних.

1.3 Аналіз мови Java

Java – це високорівнева, об’єктно-орієнтована мова програмування загального призначення. Подібно до Python, Java використовує збирач сміття для автоматичного керування пам’яттю. На відміну від Python, Java є строго статично типізованою мовою. При розробці мови було встановлено 5 цілей[3]:

- Мова повинна бути простою, об’єктно-орієнтованою та знайомою.
- Мова повинна бути безпечною.
- Мова не повинна залежити від архітектури та повинна бути портативною.
- Мова повинна бути швидкою.
- Мова повинна бути інтерпретованою та підтримувати багатопоточність.

Код програм, написаних на Java потребує компіляції у байтовий код для віртуальної машини Java. Існує багато імплементацій віртуальної машини Java для різних платформ, що надає змогу розробникам ‘написати лише раз, виконати де завгодно’[4]. Виконання байтового коду у віртуальній машині є також значно ефективнішим ніж інтерпретація коду, тому програми, написані на Java працюють значно швидше ніж відповідні програми написані на інтерпретованій мові.

```
public class Stuff
{
    public static String fibonacci(int s)
    {
        if (s > 1)
        {
            return fibonacci(s - 1) + fibonacci(s - 2);
        }
        else
        {
            return s;
        }
    }
    public static void main(String[] args)
    {
        print(fibonacci(6)); //Also prints "8"!
    }
}
```

Синтаксис Java є схожим на синтаксис C++: фігурні дужки позначають блоки коду, крапка з комою — розподільник виразів, а екларація змінної вимагає опис її типу перед назвою. На відміну від C++, у якому також є виражений синтаксис для структурованого та процедурного програмування, синтаксичні правила у Java розроблені виключно для об'єктно-орієнтованого програмування. Усі методи та статичні функції виражені всередині класів. Більшість даних є представлені об'єктами (окрім примітивних типів). Java не підтримує перевантаження операторів та множинне успадкування, але класи можуть реалізовувати довільну кількість інтерфейсів. Структура коду програми, написаної на мові Java:

- Програма складається зі списку публічних класів. Перед виконанням, у виконавчому середовищі розробник повинен вказати який клас є головним. Головний клас — це клас, який реалізовує метод "main". Цей метод є точкою входу програми.
- Кожен клас складається з переліку відкритих та приватних методів та членів класу. Методи складаються з назви, довільної кількості оголошень змінних аргументів та тіла методу. Тіло методу є блоком. Членами класу є оголошення змінних, які є доступними для кожного методу класу (окрім статичних методів)

- Блоки у Java визначаються фігурними дужками. Речення блоку розташовані між фігурними дужками та розділені роздільником. Роздільник речень у Java – це символ крапки з комою (;)
- Реченням у Java може бути оголошення змінної, присвоєння їй значення, виклик методу або розгалуження. У Java є синтаксичний вийняток що до розгалужень у реченнях: після їхнього блоку не потрібно вказувати роздільник.
- Вирази у Java мають подібну семантику до виразів з Python, та використовуються у однакових випадках.
- Змінні у Java повинні бути визначеними у блоці, в якому вони використовуватимуться. Визначення змінної відбувається вказавши її тип та назву, а також присвоївши їй значення відповідного типу. Після визначення змінної вказувати її тип у майбутніх присвоєннях значень не є необхідно. Java – це статично типізована мова. Після оголошення типу змінної, змінити його не можна.

1.4 Відмінності між Python та Java

Java та Python були розроблені дотримуючись різних філософій та цілей, вони мають ряд відмінностей між собою.

- Однією різницею між двома мовами є типізація. Java є статично-типізованою мовою: тип кожної оголошеної змінної повинен бути відомим у момент компіляції, тому його необхідно вказати у коді програми. Змінні у Java також не можуть змінювати свій тип під час виконання програми. У Python, тип змінної визначається при присвоєнні їй значення, під час роботи програми, і також може змінитися в будь-який момент.
- Python має підтримку для багатьох парадигм програмування, тому містить такі структури як функції першого класу, лямбда вирази. У

Python відсутні інтерфейси, але є підтримка множинного успадкування та ‘качиної типізації’. Python також має підтримку перевантаження операторів та засоби для метапрограмування.

- У Java є поняття головного класу: клас, який реалізовує метод main(), з якого починається робота програми. У Python потрібно вказати лише початковий файл, з якого розпочнеться інтерпретація програми. Класи та функції у Python будуються під час інтерпретації програми.
- Стандартні бібліотеки Java та Python сильно відрізняються у їхній структурі, функціоналу та можливостях.

При розробці програмного застосунку, побудованого у цій роботі, було враховано усі вказані відмінності між двома мовами.

2. Реалізація транспілятора

Результатом даної роботи є програмний застосунок для командного рядку. Виконання даного застосунку відбувається виконанням скрипта “transpile.bat”, який завантажує необхідний скрипт та передає усі задані аргументи у нього. Запуск скрипта потрібно виконувати за наступним шаблоном:

transpile <input_language> <output_language> <path_to_source_file>

, де:

- <input_language> - назва вхідної мови
- <output_language> - назва вихідної мови
- <path_to_source_file> - шлях до головного файлу вихідного коду, або перелік шляхів до усіх файлів вихідного коду.

Після роботи програми, у директорії з вказаними файлами буде створено файл (або декілька файлів) з вихідним кодом, написаним на вказаній вихідній

мові. Назви файлів будуть створені за наступним шаблоном (якщо це не суперечитиме семантиці вихідної мови):

<original_name>.<original_extension>.<output_language_extension>

2.1 Загальний огляд транспілятора

Робота транспілятора складається з усіх кроків транспіляції для будь якої підтримуваної мови. На першому кроці роботи програма зчитує послідовність символів та будує колекцію послідовностей токенів. Кожна послідовність токенів створюється за граматичними правилами вхідної мови. Наступним кроком роботи є парсинг усіх послідовностей. Результатом парсингу є АСД вхідної мови, за яким на наступному кроці буде побудовано АСД проміжного представлення програми. На останньому кроці з АСД проміжного представлення створюється послідовність символів вихідного коду на вихідній мові. Результуюча послідовність записується у вихідний файл.

Для кожної підтримуваної мови визначено пакети програмних компонентів, які реєструються у системі на початку її роботи. Ці пакети складаються з правил граматики для побудови токенів, семантичного аналізатора, перекладача АСД у проміжне представлення та генератора вихідного коду з проміжного представлення. Дана архітектура знижує час та об’єм роботи, необхідні для створення підтримки для нової мови.

2.2 Побудова токенізатора

Вхідними даними для токенізатора є послідовність символів вхідної програми та перелік граматичних правил мови. Токенізатор є скінченним автоматом, який ітеративно перетворює послідовність символів на послідовність токенів. На кожному кроці ітерації, токенізатор порівнює початок послідовності символів з переліком правил. Якщо початок послідовності відповідає певному правилу, то створюється та записується новий токен з

послідовністю символів, що відповідала правилу, а з початкової послідовності вилучається відповідна послідовнсть. Робота токенізатора припиняється тоді, коли початкова послідовність більше не містить символів, або було знайдено послідовність символів, якій не відповідає жодне правило. В останньому випадку, програма повідомляє користувача про невідому послідовність та робота програми завершується аварійно. Після виконання роботи токенізатор додає до послідовності токенів особливий токен, що позначає кінець послідовності, та повертає всю послідовність.

Токени — це елементарні синтаксичні одиниці. Вони мають тип та утримують дані про послідовність та її позицію у файлі. Для кожного типу токена існує правило, за яким виконується пошук токену серед послідовності. Правила складаються з типу токену та регулярного виразу, якому повинен відповідати токен. Правила передаються у токенізатор впорядкованою послідовністю, оскільки перевірка на відповідність правилу відбувається у визначеному порядку. Правилам на початку послідовності надається більший пріоритет ніж правилам у кінці послідовності. Це є важливим аспектом послідовності, оскільки це надає змогу визначити декілька правил, у яких множини символів, що відповідають цим правилам, можуть перетинатися. Наприклад правило для ідентифікаторів визначає послідовність типу ідентифікатора як послідовність буквено-цифрових символів довільного розміру, тоді як правило для ключових слів визначає послідовність типу ідентифікатора як послідовність, що відповідає одній з попередньо визначених послідовностей символів (“if”, “else”, “while”, тд). Очевидно, що якщо правило ідентифікаторів матиме більший пріоритет, то ключові слова буде неможливо знайти, оскільки кожне ключове слово відповідає правилу послідовності ідентифікатора.

Перелік типів послідовностей для Python та їх правил, визначений у порядку спадання пріоритету:

1. “newline” - послідовність, що складається з символів нових рядків тексту;

2. “space” - послідовність, що складається з пробілів;
3. “control” - послідовність, що складається з попередньо визначених ключових слів: “if”, “else”, “elif”, “while”, “for”, “in”, “return”;
4. “definition” - послідовність, що складається з попередньо визначених ключових слів для визначень функцій та класів: “def”, “class”;
5. “identifier” - послідовність, довільного розміру, що складається з буквено-цифрових символів, перший символ якої є літерою;
6. “comma” - послідовність, що складається з одного символу коми;
7. “integer” - послідовність, довільного розміру, що складається з цифрових символів;
8. “dot” - послідовність, що складається з одного символу крапки;
9. “colon” - послідовність, що складається з одного символу двокрапки;
10. “string” - послідовність найменшого розміру, перший та останній символи якої можуть бути символи лапок
11. “op” - послідовність, що складається з одного символу відкритої дужки;
12. “cp” - послідовність, що складається з одного символу закритої дужки;
13. “sqop” - послідовність, що складається з одного символу відкритої квадратної дужки;
14. “sqcp” - послідовність, що складається з одного символу закритої квадратної дужки;
15. “operator” - послідовність, що складається з попередньо визначених операторів: ‘+’, ‘-’, ‘*’, ‘/’, ‘!=’, ‘==’, ‘>’, ‘<’, ‘=’
16. “comment” - найкоротша послідовність, першим символом якої є ‘#’, а останнім — новий рядок. Цей тип ігнорується парсером.

Перелік послідовностей для Java:

1. “semicolon” - послідовність, що складається з одного символу закритої
2. “space” - послідовність, що складається з пробілів. Цей тип ігнорується парсером;
3. “control” - послідовність, що складається з попередньо визначених ключових слів: “if”, “else”, “while”, “for”, “return”;
4. “declaration” - послідовність, що складається з попередньо визначених ключових слів для визначень: “final”, “static”;
5. “definition” - послідовність, що складається з попередньо визначених ключових слів для визначень функцій та класів: “public”, “private”, “class”;
6. “identifier” - послідовність, довільного розміру, що складається з буквено-цифрових символів, перший символ якої є літерою;
7. “comma” - послідовність, що складається з одного символу коми;
8. “integer” - послідовність, довільного розміру, що складається з цифрових символів;
9. “dot” - послідовність, що складається з одного символу крапки;
10. “string” - послідовність найменшого розміру, перший та останній символи якої можуть бути символи лапок
11. “op” - послідовність, що складається з одного символу відкритої дужки;
12. “cp” - послідовність, що складається з одного символу закритої дужки;
13. “sqop” - послідовність, що складається з одного символу відкритої квадратної дужки;
14. “sqcp” - послідовність, що складається з одного символу закритої квадратної дужки;

15. “ob” - послідовність, що складається з одного символу відкритої фігурної дужки;
16. “cb” - послідовність, що складається з одного символу закритої фігурної дужки;
17. “operator” - послідовність, що складається з попередньо визначених операторів: ‘+’, ‘-’, ‘*’, ‘/’, ‘==’, ‘!=’, ‘>’, ‘<’, ‘=’
18. “comment” - найкоротша послідовність, першим символом якої є ‘//’, а останнім — новий рядок

2.3 Парсинг

Після побудови послідовності токенів, потрібно провести синтаксичний аналіз та побудувати абстрактне синтаксичне дерево. Побудова АСД відбувається шляхом структурованої побудови його вершин. Результуюче АСД складається з простих, складних та складених. Прості вершини будуються шляхом зчитування даних з послідовності токенів. Складні вершини будуються шляхом зчитування даних та викликом функцій для побудови похідних вершин. Складені вершини складаються з довільної кількості похідних вершин. Побудова складеної вершини відбувається шляхом визначення наступного типу вершини та викликом відповідної процедури для побудови вершини.

Для отримання токену, послідовність токенів перетворюється на чергу. Отримати токен можна тільки з голови черги, що видаляє його з черги. Також можна отримати елемент без його видалення з черги, але для правильної роботи парсера, побудова кожної вершини АСД повинна отримати усі необхідні їй токени. Залишок токенів, або їх нестача при побудові наступної вершини може привести до аварійного завершення роботи програми, або до невідповідності у семантиці результуючої програми.

Перелік типів вершин АСД для мови Python:

- IdentifierNode – вершина, що містить ідентифікаційні дані. Вона використовується у вершинах змінних, викликів функцій, функцій та класів
- ConstantNode – відображає константу, її тип та значення. Може використовуватись у виразах.
- CallNode – відображає виклик функції та послідовність її агрументів. Аргументом може бути будь-яка вершина, яку можна використовувати у виразах. Може використовуватись у виразах та реченнях.
- VariableNode – відображає змінну. Вершини змінних та викликів функцій також слугують зв’язним списком інших вершин змінних та викликів. Ця структура описує доступ до членів об’єкту або виклику методів об’єкту. Може використовуватись у виразах та реченнях.
- ExpressionNode – відображає вираз. Використовується для групування виразів. Може використовуватись у виразах.
- BinaryOperatorNode – відображає бінарний оператор. Складається з лівого операнду, правого операнду та символу оператора. Може використовуватись у виразах.
- AssignmentNode – відображає процедуру присвоєння значення змінній. Складається з вершини змінної та будь-якої вершини, яка може використовуватись у виразі. Може використовуватись у реченнях.
- ReturnNode – вершина повернення значення з функції. Складається з будь-якої вершини, що може використовуватись у виразах. Може використовуватись у реченнях.
- BlockNode – вершина, що визначає блок коду. Складається з довільного впорядкованого списку вершин, що можуть використовуватись у реченнях.

- IfBlockNode – визначає розгалуження типу “if-else”. Складається з виразу для умови, блоку розгалуження “if” та необов’язкового блоку “else”. Розгалуження типу “if-elif-else” відображаються розміщенням блоку “elif” як блок “if-else” у блоці “else”.
- ForBlockNode – визначає розгалуження типу “for-in”. Складається з ідентифікаторів для ітератора, ітерабельного виразу та блоку.
- WhileBlockNode - визначає розгалуження типу “while”. Складається з ідентифікатора для ітератора, ідентифікатора для ітерабельного виразу, та блоку.
- FunctionNode – відображає визначення функції. Складається з ідентифікатора функції, списку ідентифікаторів змінних та блоку тіла функуції.
- ClassNode - відображає визначення класу. Складається з списку вершин функуцій та присвоєнь.

Більшість типів вершин Java є подібними до типів вершин Python. Перелік типів вершин, відмінних від аналогічних або відсутніх типів вершин у Python:

- VariableNode – відображає змінну. Складається з ідентифікатора змінної та її типу.
- DeclarationNode – відображає оголошення змінної. Складається з змінної та необов’язкового виразу, який присвоюється даній змінній. Тип виразу повинен відповідати типу змінної.
- ForBlockNode – визначає розгалуження типу “for”. Складається з оголошення змінної ітерації, виразу умови виконання, виразу кроку виконання та блоку.
- MethodNode - відображає визначення методу. Складається з ідентифікатора методу, аспектів методу, списку визначень змінних, блоку

тіла методу та ідентифікатора типу, який повертає метод. Вершини цього типу можуть знаходитись тільки у вершинах класів.

- ClassNode - відображає визначення класу. Складається з ідентифікатора класу, аспектів класу, списку відкритих та приватних оголошень змінних та функцій. Вершини цього типу можуть знаходитись тільки у вершині програми.
- ProgramNode – відображає структуру програми. Складається з послідовності класів та головного класу.

2.4 Побудова проміжного представлення програми

Отримавши АСД для програми на вхідній мові, його потрібно перетворити на АСД для проміжного представлення. Побудова АСД проміжного представлення відбувається шляхом побудови вершин АСД проміжного представлення під час проходу АСД програми в глибину. Типи вершин АСД проміжного представлення було визначено максимально близькими до типів вершин у Java: IdentifierNode, ConstantNode, CallNode, VariableNode, ExpressionNode, DeclarationNode, BinaryOperatorNode, AssignmentNode, ReturnNode, BlockNode, IfBlockNode, ForBlockNode, ForInBlockNode, WhileBlockNode, FunctionNode, ClassNode та ProgramNode є ідентичними до типів вершин Java. Причиною цього є велика описовість програм, написаних на Java. За рахунок цього, АСД програми Java містить найбільшу кількість вершин серед усіх підтримуваних мов, що значно зменшує складність побудови вихідного коду з АСД проміжного представлення.

Для побудови АСД проміжного представлення з АСД програми, написаної на мові Python, потрібно:

- Описати викликані функції/використані класи з стандартної бібліотеки Python у АСД проміжного представлення;
- Визначити що виконуватиметься у головному методі головного класу;

- Визначити типи усіх змінних та функцій.

Описати викликані функції та класи з стандартної бібліотеки для будь-якої мови можна власноруч створивши їх статичні класи та методи у АСД проміжного представлення. Після цього, кожний випадок використання даних функцій або класів потрібно замінити на створені відповідні функції та класи з АСД.

У Python точка входу є головним блоком програми. У головному блоці також визначаються функції та класи програми. Кожна функція яка не є методом класу, буде реалізована у головному класі. Якщо змінні у головному блоці використовуються всередині функції, то такі змінні потрібно перетворити на члени головного класу. АСД надалі будується враховуючи задані зміни.

Визначення типів змінних відбувається шляхом визначенням типів виразів, які присвоєно цим змінним. Якщо змінній у блоці, у якому вона була оголошена, було присвоєно значення типу, відмінного від присвоєного, то створюється нова змінна з новим типом та зміненим ідентифікатором, яка продовжує використовуватись у заданому блоці. Визначення типів агрументів для функцій відбувається наступним шляхом:

1. Усі визначення функцій зберігаються як шаблони функцій.
2. Під час побудови вершини виклику функції потрібно визначити типи її аргументів. Далі, потрібно побудувати АСД даної функції за її шаблоном, вказавши визначені типи аргументів. Після цього, потрібно створити вершину виклику створеної функції та її аргументів.

Побудова АСД проміжного представлення з АСД мови Java не вимагає додаткової обробки під час проходження, оскільки АСД проміжного представлення та мови Java є подібні.

2.5 Побудова вихідного коду програми

Створивши АСД проміжного представлення, побудова послідовності символів вихідного коду програми перетворюється на тривіальну задачу. Подібно до перекладу АСД у проміжне представлення, для побудови послідовності символів необхідно виконати прохід дерева проміжного представлення в глибину. При проході кожної вершини потрібно побудувати послідовність символів, що відповідає семантиці даної вершини у вихідній мові. Якщо вершина та її похідні не мають семантичного значення у вихідній мові, то вона ігнорується. Результатом побудови є послідовність (або послідовності) символів вихідного коду програми, написаного на вихідній мові програмування. Ця послідовність записується у вихідний файл, готовий до використання у програмному середовищі вихідної мови.

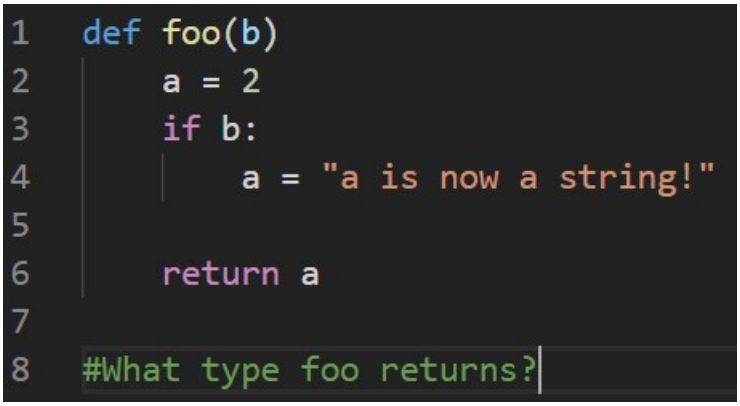
2.6 Заплановані особливості та функції

Оскільки результуюча програма не є повністю завершеною у момент створення цієї роботи, існує запланований набір особливостей та функціоналу для транспілятора:

- Повноцінний переклад та підтримка класів та успадкування
- Підтримка перекладу виключень (“try-catch”)
- Емуляція приватних методів класів у мові Python
- Підтримка лямбда-виразів у мові Java та проміжному представленні
- Дозвіл на перевизначення типу зміної у блоці, який її створив

2.7 Обмеження

На жаль, не усі аспекти мов програмування можна транспілювати між собою. Тому існують обмеження на структури та код, який можливо використовувати у транспіляторі. Одним таким обмеженням є обмеження на перевизначення типу змінних у Python.



Оскільки Python є динамічно типізованою мовою, розробник може присвоїти значення будь-якого типу змінній, яка уже утримує значення іншого типу. Таку поведінку можна емулювати у проміжному представленні оголосивши нову змінну відповідного типу, що приведе до семантично-подібної програми при експорті. Проблеми настають тоді, коли тип змінної змінюється значення, невідомого на етапі перекладу. Наприклад:

Транспілятор не зможе перекласти дану функцію у проміжне представлення та зберегти її семантичне значення. Оскільки у блоці “if” змінній “a” буде присвоєно значення типу “string”, коли її початкове значення мало тип “integer”, то транспілятор створить нову змінну, що відповідатиме новому значенню змінної “a” типу “string”. При закритті блоку, змінна “a” типу “string” стане недоступною, тому з функції завжди повертатиметься значення змінної “a” типу “integer”, тоді як у Python повертатиметься останнє присвоєне значення змінній “a”, незалежно від типу.

```
1  def foo(b)
2      if b:
3          return 42
4      else:
5          return "This will not work!"
```

У даному прикладі робота транспілятора аварійно зупиниться, оскільки при побудові функцій виконується перевірка типів, які повертає функція. Якщо у всіх “return” виразах функції цей тип не співпадає, то програма припинить роботу та повідомить про це користувача.

```
1  public class Stuff
2  {
3      private int cant_do_this_in_python;
4  }
```

Мова Python не має підтримки інкапсуляції членів класів, тож усі перетворені члени класів з програми Java будуть відкритими. Приватні методи класів можна емулювати шляхом їх визначень поза класом.

3. Опис використаних технологій

Даний проект був розроблений на мові Lua 5.3 без використання будь-яких бібліотек від третіх осіб.

Висновки

При виконанні даної роботи було визначено процес транспіляції. Було досліджено особливості мов програмування Java та Python, та їх відмінності. З використанням отриманої інформації було розроблено програмний застосунок. Результуюча програма є транспілятором між мовами програмування Java та Python. У даній роботі були описані кроки роботи цієї програми, їх особливості та обмеження. Деякі аспекти перекладу не було враховано, тому результуюча

програма не є завершеною. Нереалізовані особливості програми було описано у розділі ‘Заплановані особливості та функції’. Також було наведено обмеження на програми, які може обробити транспілятор, та вказано причини існування таких обмежень.

Використані джерела

1. https://www.eclipse.org/articles/Article-JavaCodeManipulation_AST/index.html
2. https://www.python.org/dev/peps/pep-0020/
3. https://www.oracle.com/technetwork/java/intro-141325.html
4. https://www.computerweekly.com/feature/Write-once-run-anywhere