**Зміст**

1. Вступ
2. План проходження преддипломної бакалаврської практики
3. Знайомство з підприємством
   1. Коротка історія Luxoft
4. Знайомство з проектом
   1. Знайомство з інструментарієм розробника
   2. Аналіз робочого процессу.
5. Збір матеріалів та підготовка до написання БКР
   1. Збір та знайомство з науково-методологічною літературою.
   2. Проектування конвертора
      1. Проектування лексичного алазізатора
      2. Проетування синтаксичного аналізатора
      3. Проектування етапу кодогенерації
6. Щоденник проходження практики.
7. Висновки
8. Перелік використаної літератури

**Вступ**

Вступ до пояснювальної записки переддипломної практики.

**План проходження преддипломної бакалаврської практики**

**Знайомство з підприємством**

Luxoft – міжнародна компанія що є постачальником послуг по розробці програмного забезпечення. До послуг компанії можно включити: розробку та підтримку ПЗ, проектування та тестування продуктів, технологічний консалтинг. Клієнти компанії - Boeing, IBM, Deutsche Bank, UBS, Harman, Avaya, Alstom, Sabre, Ford, Hotwire, Daimler та інші. Штат – більше 10 000 чоловік(Luxoft Ukrainе близько 3800, станом на липень 2017). Офіси компанії розташовані у східній, центральній та північній Європі, північній Америці, та південно-східній Азіі. До найбільших центрів розробки можно віднести підрозділи в: Москві, Санкт-Петербурзі, Києві, Берліні, Мюнхені та Кракові. Також Luxoft має власний освітній центр. Центр було сворено у 2000 році для професійного розвитку співробітників компанії. З 2007 року також надає освітні послуги підготовки спеціалістів по розробці програмного забезпечення з метою підвищення їх професійного рівня.

**Коротка історія Luxoft**

Компанія була заснована в Москві в 2000 році.

Ключові дати:

* 2000 — Luxoft відкриває перший американский офис в Montvale, CA
* 2001 — Luxoft відкриває центри в Сіетлі, WA, а також в Омске, Россія
* 2004 — Luxoft відкриває офіси в Лондоне, Великобританія, и Санкт-Петербурзі
* 2005 — Luxoft відкриває офіс в Києві
* 2006 — Luxoft відкриває офіс в Одессі. Отримує статус: *Microsoft Gold Certified Partner.*
* 2007 — Luxoft відкриває новий офіс в Днепропетровську, Україна
* 2008 — Luxoft відкриває офіс в Хошимиіні, В’етнам.
* 2010 — Luxoft відкриває офіс в Кракові.
* 2013 — Luxoft вийшов на IPO. Акціі торгуються на NYSE
* 2014 — Luxoft відкриває офіси в Софіі, Болгария, в Детройті, США, та в Гвадалахаре, Мексика.
* 2015 — Luxoft купує компанію Excelian (Великобританія)
* 2016 — Luxoft купує компанію InSys (США)
* 2017 — Luxoft відкриває офіс в Нижньому Новгороді.

**Знайомство з проектом**

**Знайомство з інструментарієм розробника**

**Налаштування робочого оточення розробника, завантаження та збірка проекта. Перший комміт.**

**Аналіз робочого процессу**

**Збір матеріалів та підготовка до написання БКР**

Завданням на бакалаврську дипломну роботу є розробка конвертора з високорівневої мови програмування у AutoLisp скрипт. AutoLisp –один з діалектів мови програмування LISP. Використовується у AutoCAD.

AutoCAD – комерційна система автоматизованого проектування розроблена компанією Autodesk

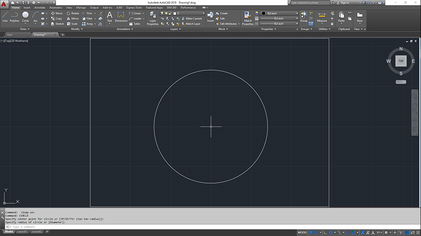


Рис Вікно користувацько інтерфейсу AutoCAD.

Перший реліз відбувся у грудні 1982 року. До AutoCAD більшість комерційних систем проектування запускались тільки на мейнфреймах, саме AutoCAD був першим що надав можливості користуватися ним на дескопномі комп’ютері. Почнаючи з 2010 система доступна також як веб- та мобільний додаток. AutoCAD широко використовується архітекторами, інженерами, графічними дизайнерами та іншими.

AutoLISP – компактний, динамічно типизиваним діалект LISP-а, у ньому відсутні сучасні потужні можливості LISP, такі як система макросів, структури. Основний акцент зроблено на роботі з геометричними примітивами(точки радіуси, кольори, кулі, лінії) та геометричними перетвореннями над ними. Також є можливість вводити координати та взаємодіяти з програмою на етапі виконання за допомогою базових функцій. Також AutoLISP має можливість створювати графічний інтерфейс користувача, та керувати діалогами для створення інтерактивних форм.

Мова значною мірою успаткувала XLISP який було сворено Девідом Бетцем. AutoLisp з’явився в AutoCAD починаючи з версії 2.18 після цього розробка була призупинена на користь більш розповсюджених технологій(VBA, .NET, ObjectARX). Але не дивлячись на це AutoLISP залишився основною мовою скриптування в AutoCAD.

Наразі також використовується VisualLISP, що являє собою значною мірою посилену версію AutoLISP-a. Ця технологія включає в себе IDE, відлагольжувальник, компілятор та підтримку технології ActiveX. Технологія розповсюджується як аддон, починаючи з 14 березня 1994. AutoLISP достатньо популярна технологія, стороні розробники САПР([Bricscad](https://en.wikipedia.org/wiki/Bricscad), [IntelliCAD](https://en.wikipedia.org/wiki/IntelliCAD_Technology_Consortium)**)** додали її підтримку до своїх продуктів.

Розробку технічної частини було вирішено писати на мові програмування Python 3.6. Python – високорівнева мова програмування загального призначення орієнтована на підвищення продуктивності розробника. Основні риси:

* Динамічна типізація;
* Автоматичне керування пам’яттю;
* Повна інтроспекція коду;
* Механізм обробки виключних ситуацій
* Підтримка багатопоточних розрахунків
* Зручні структури даних.

Однією з ключових рис Python-a що буде корисної при виконанні дипломного проекту є підтримка регулярних виразів. Регулярні вирази – формальна мова пошуку підстрок в тексті, що основана на використанні мета-символів. Для пошуку використовутся строка-шаблон яка складається зі символів та мета-символів, задаючих правило пошуку.

Наприклад регулярниій вираз що описує будь-яку послідовність з п’яти чи більше десяткових цифр виглядає наступним чином:

[0-9]{5,}

Або більш коротка версія:

\d{5,}

Де:

\d – мета-символ що позначає цифру від 0 до 9

Синтаксис регулярних виразів Python успадкований у мови програмування Perl.

**Збір та знайомство з науково-методологічною літературою.**

Готуючись до виконання дипломної роботи я зібрав матеріал що освітлює питання проектування та реалізації транслятора.

Найбільш популярною книгою в якій розповідається про трансляцію мов програмування – є «Компілятори: принципи, технології та інструменти», більш відома як «книга дракона». В ній широко освітлюється етапи лексичного, синтаксичного, семантичного аналіза, обговорюються етапи генерації проміжного та цільового коду та оптимізації. На відмону від більшості схожих книг містить багато прикладів коду, починаючи з простого одопрохідного компілятора до більш складних конструкцій.

Більш теоретично направлею є книга «Введення в теорію автоматів, мов та розрахунків» Дж. Хопкрофта, Р. Мотвани, Дж Ульмана. Книга відомих американських вчених що присвячена теорії автоматів, відповідних формальних мов та граматик – як регулярних так і контексно – незалежних. Автори детально розглядають машини Тьюринга, з допомогою яких формалізуються поняття здійсненних танездійсненних проблем, а також визначаються функції часової та ємнісної оцінки складності алгоритмів. Текст підручника супроводжується детальними прикладами, а також задачами для самостійного розвитку.

Мові програмування Python присвяченно багато статей та книг, але потрібно виділити «Програмування на Pyton» М. Лутца. В ній розповідається про затосування мови Python для вирішення типових задач зо виникають в різних прикладних областях. Освтлені теми від сворення графічного інтерфейса до програмування веб-додатків і досліджуються прийоми програмування мережевих взаємодій, взаємодій з базами даних, обробки тексту, створення інтерфейсів для сценаріїв і в багатьох інших областях. Незважаючи на те, що протягом всієї книги використовується мова Python, проте основна увага буде приділятися не основа мови, а прийомам вирішення практичних завдань. Також книга ставить собі за мету познайомити читача з часто використовуваними інструментами і бібліотеками. Таким чином, дана книга є ресурсом, що дозволяє читачеві отримати більш глибоке розуміння ролі мови Python в практиці програмування.

Книга Джеффри Фридла «Регулярні вирази» зосереджена на написанні високопродуктивних регулярних виразів. Розглядаються приципи дії механізма регулярних виразів, порівнюються функціональні можливості різних мов програмування і інструментальних засобів. Ретельно досліджується питання оптимізації, навчає правильно конструювати регулярні вирази для самих різних ситуацій. Також автор показує набільш загальні помилки при використанні регулярних виразів і допомагає їх уникнути.

**Проектування конвертора**

Конвертор – программа що транслює вихідний код однії високорівневої мови у вихідний код іншой високорівневої. Транслятор зазвичай виконує також діагностику помилок, формує словники ідентифікаторів, видає для друку текст програми. Мова, на якому представлена вхідна програма, називається вихідним мовою, а сама програма - вихідним кодом. Вихідний мова називається цільовим мовою, а вихідна (результуюча) програма - об'єктним кодом.

Існують наступні види трансляторів:

* Діалоговий транслятор – забезпечує використання мови програмування в режимі розподілення часу.
* Синтаксично-орієнтований транслятор – транслятор що тримую на вхід опис синтаксису та семнтики мови програмування та вихідний текст і виконує трансляцію згідно опису.
* Однопрохідний транслятор – створює об'єктний модуль при однократному послідовному читані вихідного кода
* Багатопрохідний транслятор – створює об'єктний модуль після декількох етапів читання вихідного кода.
* Оптимізуючий транслятор – виконує оптимізацію утворюваного коду.
* Зворотній транслятор – виконує перетворення цільового коду на вихідний код.

В ході виконання дипломної роботи буде реалізован простий онопрохідний транслятор.

Процес трансляції складається з наступних етапів:

* Лексичний аналіз. Транслятор на цьому етапі перетворює вихідний код на послідовність лексем
* Синтаксичний (граматичний) аналіз. На початку цього етапу транслятор отримує послідовність лексем яку він перетворить у абстрактне синтаксичне дерево.
* Семантичний аналіз. Під час семантичного аналізу транслятор обробляє синтаксичне дерево з метою встановлення його семантики. Зазвичай це може бути прив’язка ідентифікаторів до іх декларацій, встановлення типів змінних, визначення типів виразів. Результатом роботи цього етапу трансляції є «проміжній код». Часто проміжний код розробники доповнюють допоміжною службовою інформацією для полегшення подальшої обробки.
* Оптимізація. На цьому етапі транслятор займається видаленням зайвих операції, при цьому зберігаючи семантику вихідного кода, створеного программістом. Як правило, вхідними даними для цього етапу є проміжний код отриманий на попередьоиу етапі, але також зустрічається оптимізація коду цільової машини.
* Генерація коду. На цьому етапі транслятор перетворює проміжний код на код цільової мови.

Вихідна мова представляє собою декларативний спосіб представлення графічних об’єктів на кресленні. Комбінуючи вбудовані примітиви проектувальник зможе створювати складні графічні об’єкт.

Приклад вихідного кода:

MAIN {

Type {

property Int x: 0;

property Point top: Point { x: 90; y: 78 };

Line { begin: x; end: top.x};

Line { begin: top.x; end: top.y; }

}

Type { x: 10; top: Point { x: 34; y: 90; }}

Type {}

}

Створить на креслені два графічних об’єкта один з дефолт ними параметрами інший з встановленими при інстанціювані типу.

**Проектування лексичного алазізатора**

Лексичним аналізом називають процес аналітичного розбору вхідної послідовності символів на розпізнані групи - лексеми, з метою отримання на виході ідентифікованих послідовностей, званих «токенами» (подібно до угрупованню букв в словах). У простих випадках поняття «лексема» і «токен» ідентичні, але більш складні токенізатори додатково класифікують лексеми по різним типам ( «ідентифікатор, оператор», «частина мови»). Лексичний аналіз використовується в компіляторах і інтерпретатора вихідного коду мов програмування, і в різних парсером слів природних мов.

Опишемо існючи лексеми у вихіній мові за допомогою регулярних виразів, поставивши у відповідність кожному шаблону тег, який опише тип до якого відноситься лексема:

* ("\d+\.\d+", Tag.REAL) – число з плаваючою точкою
* (\d+, Tag.INTEGER) – ціле число
* ((property)|(import)|(alias)|(name), Tag.KEYWORD ) - зарезервоване мовою в службових цілях слово
* ([A-Z][a-z0-9]+, Tag.TYPENAME) – валідне им’я типу
* (r"[a-z]+[0-9]\*", Tag.NAME) – ім’я «властивості» типу
* ,(r"\+|-|\\*|/", Tag.OPERATOR), - арифметичний оператор
* (r"\s+", Tag.SEPARATOR), - символ що розділяє лексеми
* (r";", Tag.SEMICOLON), - точка зап’ята, служить маркером кінця інструкції
* (r":", Tag.COLON), - двокрапка служить оператором присвоєння значення
* (r"\n", Tag.EOL), - кінець строки
* (r"\(", Tag.OPEN), - відкриваюча дужка
* (r"\)", Tag.CLOSE), - закриваюча дужка
* (r"{", Tag.BEGIN), - початок простору імен
* (r"}", Tag.END), - кінець простору імен
* (r",", Tag.COMMA), - зап’ята
* (r"\.", Tag.DOT), - точка. Оператор доступу до «властивості» об’єкту

Алгоритм роботи лексичного аналізатора наступний:

1. Поставити курсор у початок текстового файлу
2. Послідовно співставити кожний шаблон з вихідним текстом починайчи з позіції курсора
3. Якщо співставлення завершилось успішно, поставити курсор у кінець сопоставленного тексту, додати знайдену лексему до результуючого списку і перейти до кроку 2
4. Якщо жоден з шблонів не підійшов завершити виконання программи та возбудити виключну ситуацію

Приклад реалізації алгоритму на мові програмування Python:

def tokenize(self, string: str, position: int) -> list:

length = len(string)

lexems = []

while position < length:

lexem = self.findLexem(string, position)

position += lexem.length

if lexem.tag not in Lexer.ignorable:

lexems.append(lexem)

return lexems

def findLexem(self, string, position) -> Lexem:

for token in Lexer.tokens:

match = token.regex.match(string, position)

if match is not None:

return Lexem(

match.group(), token.tag,

match.start(), match.end(),

)

raise SyntaxException("error")

Приклад роботи лексичного аналізатора де вихідний код виглядає наступним чином:

"Type {\n" +

" property x: 0;\n" +

" property y: 1.01;\n" +

" Line{ start: x; end: y; };\n" +

"}\n",

Буде послідовність лексем:

Lexem("Type", Tag.TYPENAME), Lexem("{", Tag.BEGIN),

Lexem("property", Tag.KEYWORD), Lexem("x", Tag.NAME),

Lexem(":", Tag.COLON), Lexem("0", Tag.INTEGER),

Lexem(";", Tag.SEMICOLON), Lexem("property", Tag.KEYWORD),

Lexem("y", Tag.NAME), Lexem(":", Tag.COLON),

Lexem("1.01", Tag.REAL), Lexem(";", Tag.SEMICOLON),

Lexem("Line", Tag.TYPENAME), Lexem("{", Tag.BEGIN),

Lexem("start", Tag.NAME), Lexem(":", Tag.COLON),

Lexem("x", Tag.NAME), Lexem(";", Tag.SEMICOLON),

Lexem("end", Tag.NAME), Lexem(":", Tag.COLON),

Lexem("y", Tag.NAME), Lexem(";", Tag.SEMICOLON),

Lexem("}", Tag.END), Lexem(";", Tag.SEMICOLON),

Lexem("}", Tag.END),

**Проектування синтаксичного аналізатора**

Синтаксичним аналізом називають процес зіставлення лінійної послідовності лексем (слів, токенов) природного або формальної мови з його формальної граматикою. Результатом зазвичай є дерево розбору (синтаксичне дерево). Зазвичай застосовується спільно з лексичним аналізом.

Синтаксичний аналізатор - це програма або частина програми, що виконує синтаксичний аналіз.  
В ході розбору вихідний текст перетворюється в структуру даних, зазвичай - в дерево, яке відображає синтаксичну структуру вхідної послідовності і добре підходить для подальшої обробки.

Як правило, результатом синтаксичного аналізу є синтаксична будова речення, представлене або у вигляді дерева залежностей, або у вигляді дерева складових, або у вигляді деякого поєднання першого і другого способів

Типи реалізацій алгоритму синтаксичного аналізу:

* Нисхідний парсер – продукціїї граматики розкриваються, починаючи зі стартового символу, до отримання потрібної послідовності лексем (Метод рекурсивного спуска, LL-аналізатор)
* Висхідний парсер – продукції востанавлюються з правих частин, починаючи з токенів і закінчуючи першим символом(LR, GLR – парсери).

Виконуючи дипломну роботу реалізуємо простий LL аналізатор. Дана частина програми прийматиме послідовність лексем, та будуватиме з них синтаксичне дерево розбору. Для прикладу подивимось на реалізацію парсера арифметичних виразів.

Алгоритм роботи наступний:

1. Прочитати перший операнд
2. Прочитати оператор
3. Прочитати одеранд
4. Побудувати дерево

Алгоритм побудови дерева насупний:

1. Якщо лівий операнд не бінарний вираз – повернути дерево де лівий опернд – лівий опернад переданий у якості аргумента, правий – правий опернд
2. Якщо правий операнд не бінарний вираз то перейти до кроку 3
3. Якщо вага лівого оператора більша або дорівнює вазі поточного то повернути бінарне дерево аналогічне попоредьньому, інакше перейти до кроку 4
4. Повернути бінарний вираз де оператор – оператор лівого виразу, лівий операнд – лівий операнд лівого виразу, правий – бінарний вираз де оператор – поточний оператор, лівий операнд – правий операнд ливого виразу, правий – правий вираз
5. Якщо вага поточного оператора менша за вагу лівого виразу перейти до шагу 6 інакше до 7
6. Повернути бінарний вираз аналогічний до виразу на 4 кроці
7. Повернути вираз аналогічний виразу на 1 кроці.

Вихідний код на мові програмування Python:

def parse(self, lexems, position):

root, position = self.takeOperand(lexems, position)

while not self.finish(lexems, position):

operator, position = self.takeOperator(lexems, position)

right, position = self.takeOperand(lexems, position)

root = self.makeTree(operator, root, right)

return root, position + (1 if self.hasParen else 0)

def makeTree(self, operator: str, lhs, rhs):

if lhs.\_\_class\_\_ is not BinaryExptrAST:

return BinaryExptrAST(operator, lhs, rhs, self.hasParen)

currentWeight = WEIGHT[fromStr(operator)] + (10 if self.hasParen else 0)

if rhs.\_\_class\_\_ is not BinaryExptrAST:

if lhs.weight >= currentWeight:

return BinaryExptrAST(operator, lhs, rhs, self.hasParen)

else:

return BinaryExptrAST(lhs.operator, lhs.lhs, BinaryExptrAST(

operator, lhs.rhs, rhs, False

), self.hasParen

)

if lhs.weight < currentWeight:

temp = lhs.rhs

return BinaryExptrAST(lhs.operator, lhs.lhs,

BinaryExptrAST(operator, temp, rhs, lhs.isParen), self.hasParen

)

else:

return BinaryExptrAST(operator, lhs, rhs, self.hasParen)

**Проектування етапу кодогенерації**

Щоденник проходження практики.

Висновки

Перелік використаної літератури