УДК 004.4

Сугоняк І.І.
Житомирський державний технологічний університет

Марчук Г.В.
Житомирський державний технологічний університет

Бобровнік С.О.
Human Interface Technology

СИНТАКСИЧНИЙ АНАЛІЗ КОДУ ДЛЯ СИСТЕМИ ДИСТАНЦІЙНОГО НАВЧАННЯ ПРОГРАМУВАННЯ НА МОВІ C#

Дослідження присвячене розробці алгоритму аналізатора коду для системи дистанційного навчання мов програмування. Отриманий результат полягає у модифікації методу синтаксичного аналізу коду на основі ітеративного алгоритму. Також у роботі побудовано модель синтаксичного аналізатора з розподілом лексем. Ця модель є математичним забезпеченням програмного комплексу дистанційного навчання програмування. Окрім того, в роботі викладено підхід до реалізації візуальних розподілених навчальних комплексів із використанням COTS-архітектури.

Ключові слова: *синтаксичний аналізатор, ітеративний алгоритм, аналіз програмного коду, дистанційне навчання, C#.*

Актуальність теми. З часу впровадження перших програмованих машин було створено понад дві з половиною тисячі мов програмування і з кожним роком кількість їх збільшується. Деякими мовами вміє користуватись тільки невелике число їх власних розробників, інші стають відомі мільйонам людей. Актуальність теми дослідження зумовлена тим, що для успішного та доступного навчання мов програмування потрібно реалізувати зручний візуальний навчальний комплекс з подальшою оцінкою успішності.

Метою дослідження є аналіз та побудова алгоритмів та моделей аналізатора коду для розробки системи дистанційного навчання мов програмування з коректною автоматичною оцінкою прогресу користувача. Основними завданнями, вирішеними в дослідженні, є: проведення аналізу теоретичних засад проектування та реалізації візуального навчального комплексу; розробка математичної та алгоритмічної моделі функціонування візуального навчального комплексу; реалізація програмної системи візуального навчального комплексу «Програмування C#».

Аналіз останніх досліджень і публікацій. Дослідженню роботи компіляторів та алгоритмів синтаксичного аналізу присвячено багато досліджень у зарубіжній науці такими науковцями, як: Альфред Ахо [1], Джеффрі Ульман, Раві Сеті, Ніклаус Вірт [2], Юрг Гуткнехт, Дик Грун, Робин Хантер. Слід звернути увагу на статтю [3] українського науковця В.І. Салапатова, де розглядається синтаксичний аналіз із розподілом лексем на групи. Такий підхід дає змогу значно спростити синтаксичний аналіз операторів мови та прискорити його виконання.

Отримані результати полягають у модифікації методу синтаксичного аналізу коду з використанням класичних алгоритмів та побудові моделі синтаксичного аналізатора з розподілом лексем на групи, який працює швидше, ніж класичні алгоритми через направлений перебір варіантів. Практичне значення роботи полягає у реалізації методів оцінювання прогресу навчання та синтаксичного аналізу текстів коду; розробці візуального навчального комплексу «Програмування C#» з оптимальними методами оцінювання.

Основна частина

Синтаксичний аналіз (parsing) – це процес співставлення лінійної послідовності лексем мови з його формальною граматикою. Під час синтаксичного аналізу текст оформлюється у структуру даних, зазвичай в дерево, яке відповідає синтаксичній структурі вхідної послідовності і добре підходить для подальшої обробки. Найчастіше синтаксичні аналізатори працюють у два етапи: на першому ідентифікуються осмислені токени, на другому створюється дерево розбору.

Синтаксичний аналіз мови програмування заснований на контекстно-вільній граматиці, за допомогою якої можна визначити більшу частину синтаксичної структури мови програмування.

Під час дослідження розглянуто *ітеративний алгоритм*, який багаторазово виконує базові методи аналізу з уточненням результатів на кожній ітерації. Особливістю цього алгоритму є те, що він завершується за скінченне число ітерацій, а отримане рішення є повним. Ємність пам'яті, якої потребує цей метод, лінійно залежить від довжини ланцюжка, що аналізується, но час може виражатися експонентою. Серед переваг ітеративного алгоритму відзначимо можливість управляти точністю і обчислювальною складністю аналізу за рахунок обмеження числа ітерацій. Показано, що рішення, отримане на будь-який ітерації алгоритму, є повним і може використовуватися для виявлення дефектів.

Також розглянуто алгоритм *Ерлі*. Цей метод синтаксичного аналізу дає змогу для довільної КВ-граматики розібрати вхідний ланцюжок за час $O(n^3)$, використовуючи при цьому ємність пам'яті $O(n^2)$, де n – довжина вхідного ланцюжка. Якщо граматика однозначна, то час дорівнює n^2, для більшості граматик мов програмування алгоритм можна модифікувати так, щоб час та ємність стали лінійними функціями від довжини вхідного ланцюжка.

Такі алгоритми дають змогу доволі просто і коректно проводити синтаксичний аналіз коду.

Основною метою впровадження дистанційної форми навчання є швидке й зручне поширення знань, забезпечення доступності освіти всім верствам населення. Значною мірою ця мета реалізується за допомогою програмних засобів, побудованих на сучасних інформаційно-комунікаційних технологіях, які одержали загальну назву системи дистанційного навчання (СДН). До найпопулярніших СДН можна віднести: Lotus Learning Space; Blackboard Learning System; REDCLASS; «Клас ХПІ». Слід зазначити, що у всіх системах відбувається ручна перевірка написаного коду, внаслідок чого аналіз результатів є надзвичайно нечітким і може бути упередженим, що унеможливлює якісний та об'єктивний моніторинг компетентностей студента.

Моніторинг допомагає відстежувати якість засвоєних знань і вмінь у навчальному процесі. Тому основною метою розробки та впровадження візуального навчального комплексу «Програмування C#» є підвищення ефективності навчання.

Результатом реалізації поставленого завдання є розгорнутий візуальний навчальний додаток, що містить у своєму складі серверну структуру збереження та обробки даних, кросплатформенний багатокористувацький клієнтський додаток для реалізації функціональних можливостей.

Центральним складником обробки даних у навчальному додатку є синтаксичний аналіз коду виконання навчальних вправ мовою програмування C#.

Синтаксичні аналізатори, які зазвичай використовуються в компіляторах, підрозділяють на два типи: спадні (top-down) і висхідні (bottom-up). Перші будують дерево розбору зверху до низу – від кореня до листя, другі від листя до кореня дерева. В обох випадках розбір рядка відбувається зліва направо.

Існує безліч реалізацій синтаксичних аналізаторів для розбору різного роду граматик. У разі мов програмування для розбору контекстно-вільних граматик використовуються так звані LL(k) (Left Left) і LR(k) (Left Right) аналізатори.

Розглянемо на прикладі алгоритм розбору для LL(1)- граматики.

G_1: (1) S → aAS; (2) S → b; (3) A → a; (4) A → bSA

Керуюча таблиця алгоритму представлена на рисунку 1.

	a	b	e
S	aAS,1	B,2	помилка
A	A,3	bSA,4	помилка
a	викид	помилка	помилка
b	помилка	викид	помилка
$	помилка	помилка	допуск

Рис. 1. Керуюча таблиця алгоритму

За допомогою таблиці проаналізуємо такий ланцюжок abbab:

(abbab,S$,e) ├ (abbab, aAS$, 1) ├ (bbab, AS$, 1) ├ (bbab, bSAS$, 14) ├ (bab, SAS$, 14) ├ (bab, bAS$, 142) ├ (ab, AS$, 142) ├ (ab, aS$, 1423) ├ (b, S$, 1423) ├ (b, b$, 14232) ├ (e, $, 14232)

Очевидно, що 14232 – лівий розбір ланцюжка *abbab*.

Розглянуті алгоритми передбачають посимвольний перегляд тексту програми з кроком уперед. При цьому всі елементи оператора мають оброблятися за єдиним алгоритмом. Це вносить певні ускладнення, оскільки нетермінали, які мають бути присутніми у певних місцях правил граматики, описуються за своїми власними правилами. Ці правила мають вигляд:

G = f(T, N, P, Ns), де

– *Ns* – кореневий символ (спеціальний нетермінал);

– *T* – це множина термінальних символів (у мові програмування це константи, ідентифікатори, ключові слова, символи пунктуації);

– N – це множина нетермінальних символів (у нашому разі такі поняття, як вирази, оператори та окремі частки операторів).

У зв'язку з цим доречно розглядати граматику кожного нетермінала та його обробку окремо. Так, якщо згідно з граматикою у певному місці оператора має бути вираз, то обробку оператора мови в цій частині треба виконувати згідно з граматикою виразу, тобто окремим модулем. При цьому ознакою кінця виразу є або ключове слово з конструкції оператора, або спеціальні знаки-розподільники.

Поєднання в одному аналізі розбору оператора, розбору виразу та інших частин оператора ускладнює та уповільнює процес компіляції у цілому. Тому розробка нових ефективних засобів синтаксичного аналізу у мовах програмування на цей час є все ще актуальною.

Як наведено у [6], можна виділити дві великі групи лексем: лексеми-об'єкти (описують дані) та лексеми дії (дії над цими даними). Таким чином,

$$L \rightarrow Ld + La,$$

де L – множина всіх лексем, Ld – лексеми-об'єкти, La – лексеми дії.

Для вирішення поставленої проблеми використаємо модифікований алгоритм «знизу догори».

Таблиці ідентифікаторів використаємо для визначення лексем та ключових слів. Вирази описуються граматикою, яка співпадає з правилами арифметичних дій, тобто існує пріоритетність виконання операторів, урахування дужок, які змінюють пріоритетність дій, та деякі інші. Ознаками кінця виразу можуть бути як ключові слова операторів, так і спеціальні символи.

Лексичний аналізатор обробляє лексеми L_d під час їх появи, причому тип результату визначається L_a. Оскільки порядок їх обробки визначений пріоритетом, це значною мірою спрощує опис правил граматики.

В останню чергу виконується породження гілок дерева поточного оператора як результат обробки ключових слів (через найнижчий пріоритет). Стартовий символ N_s, тобто тип оператора мови, розпізнається через першу або другу лексему.

Лексеми дії мають стандартний за семантикою набір, що присутній в усіх мовах програмування, та спеціальний набір тільки для конкретної мови. Пріоритетність дій у виразах визначає порядок виконання цих дій. Якщо пріоритет поточної лексеми дії не перевищує пріоритет попередньої, то виконується обробка попередньої лексеми дії. Інакше обробка відкладається у стек. Зобразимо порядок обробки лексем за допомогою діаграми активності, зображеної на рисунку 2.

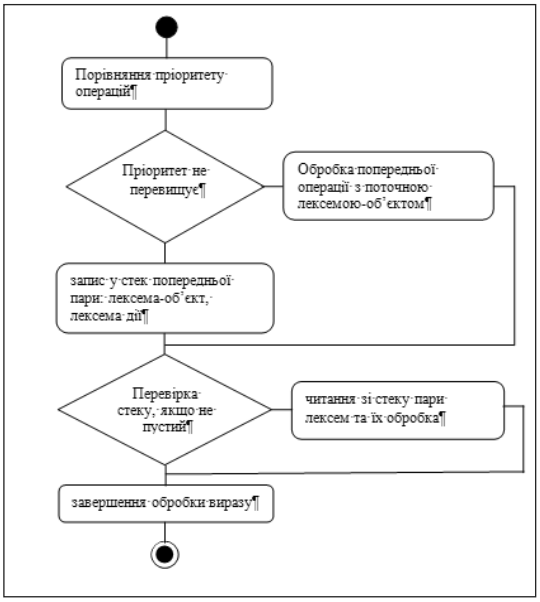

Рис. 2. Порядок обробки лексеми дій

Виконання алгоритму розподілу на лексеми реалізується механізмом доступу до стекової пам'яті. Якщо вираз має дужки, то виконання алгоритму починається із запису лівої дужки у стек з нульовим операндом і закінчується з появою правої. Права дужка ініціює зворотний хід у стеку, тобто обробку лексем у стеку з наступним їх виштовхуванням звідти. Обробка виразу закінчується, коли стек буде пустим. Результатом обробки є гілки синтаксичного дерева розбору.

Відсутність паритету дужок одразу ж виявляється через стек як помилка. У разі невідповідності типів операндів стандартний алгоритм має передбачати в разі необхідності перетворення типів (за допомогою спеціальних модулів). Тип результату виразу має відповідати опису оператора.

Розберемо вираз a + a * a для граматики G_2.

G_2: (1) E → E+T; (2) E → T; (3) T → T*F;
(4) T → F; (5) F → (E); (6) F →a.

Отже, відповідно до нашої граматики вираз a + a * a буде розпізнаний таким чином:

$(S, a + a * a, h_0) \vdash (S , a + a * a , h_0+) \vdash (S , a + a * a, h_0+) \vdash (S , a + a * a, h_0 +) \vdash (S, a + a * a, h_0 + a) \vdash (S, + a * a , h_0 +) \vdash (S, a * a, h_0) \vdash (S, a * a, h_0 *) \vdash (S, a * a, h_0 *) \vdash (S, a * a, h_0 * a) \vdash (S, * a, h_0 *) \vdash (S, a, h_0) \vdash (S, a, h_0a) \vdash (S, \$, h_0) \vdash (S, \$, \$).$

Послідовність правил, що була використана, відповідає лівому виводу вхідного ланцюжка:

$$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow a +T \Rightarrow a +T*F \Rightarrow a + F*F \Rightarrow a + a *F \Rightarrow a + a * a.$$

Дерево синтаксичного розбору буде мати вигляд, що представлений на рисунку 3. Побудова відбуватиметься зверху донизу.

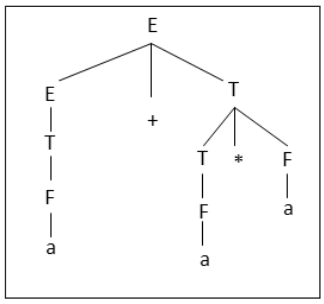

Рис. 3. Дерево синтаксичного розбору

За заданим алгоритмом опис граматики зводиться до опису ключових слів (зарезервованих і контекстних). Нині у версії 4.0 мови C # визначено 77 зарезервованих ключових слів і 18 контекстних. Оскільки лексеми дії можуть бути як спеціальними символами, так і ключовими словами, то код буде складатися з коду пріоритету та власне коду лексеми, а це певний тип обробки. Так, наприклад, ключове слово if породжує команди порівняння та відповідного умовного переходу. Всі ключові слова можна звести в одну таблицю та закодувати їх, щоб відрізнити лексеми дії виразу від лексем дії операторів.

Доречно робити розподіл обробки виразів та операторів. Окремим модулем робити обробку виразів, що на виході буде будувати дерево розбору виразу, а ключові слова операторів мови обробляти окремо.

Згідно з правилами граматики лексеми дії мають передбачати як стандартні, так і нестандартні дії. Всі дії задаються в таблиці ключових слів кодом, що є посиланням, що і забезпечує перехід на відповідну частину програми синтаксичного аналізатора.

Пошук термінала здійснюється у таблиці ключових слів, а тип нетермінала визначається типом лексеми, яка може бути оператором або виразом. Все це спрощує перевірку правильності синтаксису і дає змогу швидко виявити синтаксичні помилки.

Крім того, є стандартні оператори, такі як умовний оператор (if), оператори циклу (for, while, do… while), оператори введення/виведення, а є нестандартні (наприклад, побітові операції), до яких необхідно визначити правило обробки та додати модуль синтаксичного розбору. Звернення до них буде через посилання, яке представлено як код у таблиці ключових слів.

Ключові слова – це попередньо визначені зарезервовані ідентифікатори, які мають особливе синтаксичне значення, які мають спеціальні значення для компілятора і які мають певну послідовність в тій чи іншій конструкції оператора мови, обробка яких виконується наприкінці, тобто з найнижчим пріоритетом. Пріоритет використовується в основному при обробці виразів і досить зручно вписується у загальний алгоритм синтаксичного аналізу. У мовних конструкціях ключові слова операторів також можуть використовуватися як ознаки кінця виразу.

На етапі лексичного аналізу виконується розпізнавання не однієї лексеми, а пари лексем: лексеми-об'єкта та лексеми дії і передається далі синтаксичному аналізатору для подальшого розбору. На першому кроці аналізу оператора мови лексичний аналізатор розпізнає кореневий символ N_s, що визначає тип оператора, а подальший алгоритм розбору відтворюється згідно з правилами граматики.

Оскільки аналіз здійснюється через розбір пари лексем, то швидкість обробки операторів вища, ніж за класичними алгоритмами через направлений перебір варіантів, але ускладнюється алгоритм лексичного аналізу, а це впливає на загальну швидкість синтаксичного аналізу. Крім збільшення швидкості розбору, перевагою цього методу є спрощення правил опису граматики.

Крім того, в роботі розроблений алгоритм оцінювання прогресу студента, що базується на традиційних та нових (сучасних) методах контролю. До використаних традиційних методів контролю можна віднести тестування та практичні завдання. Якщо говорити про сучасні методи контролю, то використано кейс-вимірювачі, проекти, катанотести та контекстні завдання.

Завдання складається із декількох блоків, які допомагають студенту покращити свої знання в сфері програмування. Після проходження кожного блоку завдань студент проходить перевірку знань. Допоки студент не пройде попередній блок завдань наступний не буде доступний для навчання. Можна декілька разів проходити курс, з кожним разом покращуючи свої знання.

Блок завдань складається із декількох логічних рівнів:

- надання інформації по темі курсу;
- тестові завдання після закінчення курсу;
- практичне завдання після успішного проходження тестових завдань.

Тестові завдання додані з метою перевірки теоретичних знань студента, вони перевіряють знання основних положень курсу.

У вирішенні практичних завдань може бути декілька варіантів вирішення поставленої проблеми. Наприклад, якщо поставлено завдання від-

сортувати масив, студент може використати будь-який тип сортування, після написання алгоритму аналізаторами буде перевірено вихідний масив і виставлена оцінка.

Після успішного проходження блоку завдань студент отримує винагороду у вигляді 3D об'єкта, який можна поставити у віртуальному просторі за допомогою використання технології доповненої реальності.

Для переходу на новий рівень потрібно набрати вісімдесят чи більше відсотків успішних відповідей. Практичні і теоретичні завдання займають сімдесят та тридцять відсотків вартості відповідно.

У візуальному навчальному комплексі наявні такі теми:

– типи даних, літерали і змінні в мові програмування C#;
– оператори, масиви і рядки;
– введення в класи, об'єкти і методи;
– алгоритмічна підготовка студента з використанням псевдомови;
– наслідування, поліморфізм, інкапсуляція;
– інтерфейси, структури;
– делегати, події і LINQ;
– опрацювання виняткових ситуацій.

Під час навчання студент зможе покращити свої знання базових алгоритмів програмування, ознайомитися з Microsoft. Net Framework, C# та об'єктно орієнтованим програмуванням. Також під час програмування він розбереться з типами, зумовленими користувачем, структурами, класами, конструкторами, дізнається все про збирач сміття.

Для реалізації візуального навчального комплексу було використано COTS-архітектуру у взаємодії із об'єктно-центрованою моделлю (рис. 4). COTS – component off the shelf – методологія, яка дає розробникам можливість використання сторонніх компонентів.

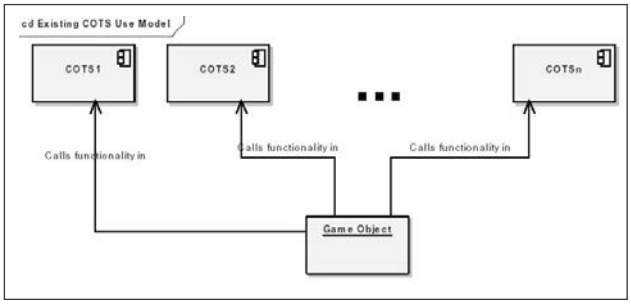

Рис. 4. COTS-архітектура у взаємодії із об'єктно-центрованою моделлю

Реалізація системи нараховує більше ста класів. У системі представлені класи, які описують операції з виведенням та проходженням завдань, графічною системою користувача, системою управління об'єктів та інше.

Для визначення спільного кордону функціональності системи та з метою створення основи

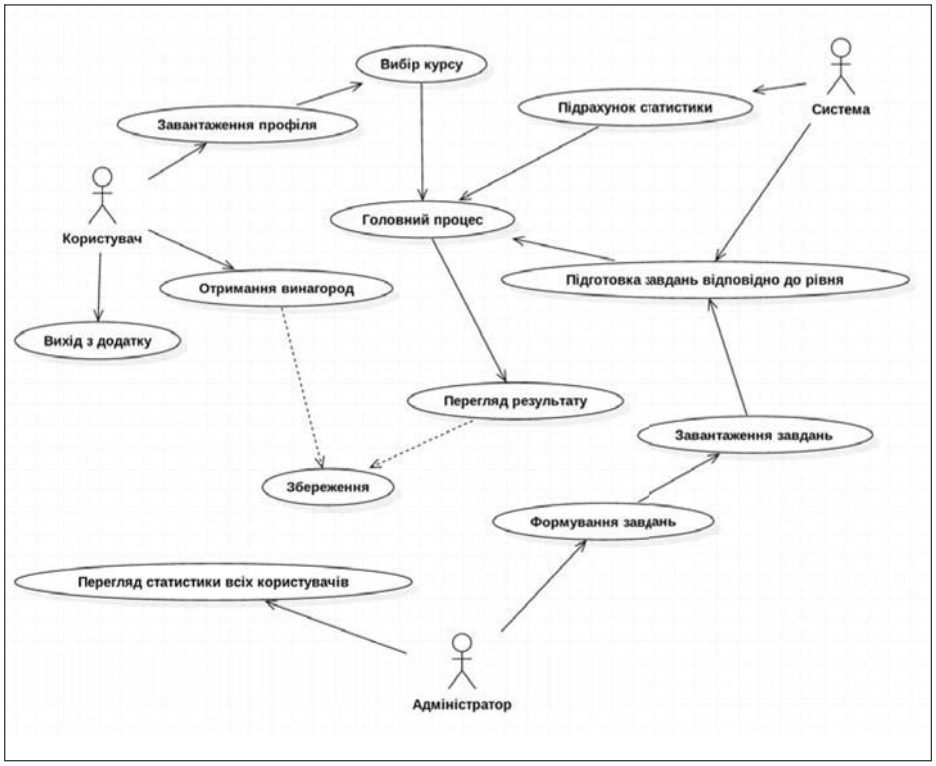

Рис. 5. Діаграма варіантів використання навчального комплексу

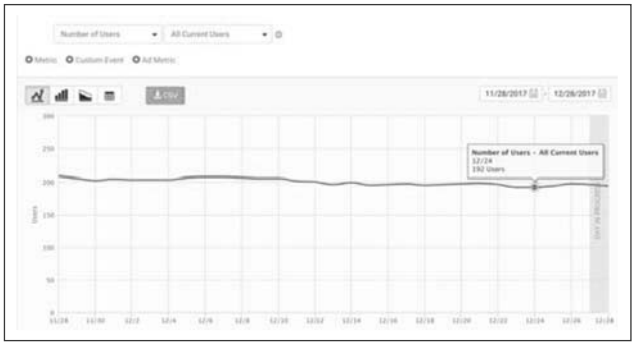

Рис. 6. Перегляд кількості активних студентів

для виконання аналізу, проектування, розробки і тестування була розроблена діаграма варіантів використання (рис. 5). Ця діаграма допомагає сформулювати загальні вимоги до функціональної поведінки проектованої системи, розробити вихідну концептуальну модель системи.

Для динамічного завантаження даних із сервера була розроблена Asset Bundle система. Така система містить у собі завантаження Asset Bundle, кешування, автоматичне визначення варіантів колекції, перезавантаження колекції зі зміненими варіантами. Цей сервер дає змогу в процесі виконання додатка автоматично завантажувати нові курси користувачу, змінювати контент та зберігати статистику навчання. Для розпізнання тексту написаний лексичний аналізатор коду, який переводить послідовність символів у групи, що відповідають певним шаблонам, з визначенням їхніх типів.

Написання програмних анімацій відбулося за допомогою бібліотеки iTween. Така бібліотека є простою у використанні та працює з усіма типами об'єктів.

Перегляд статистики результатів навчання відбувається через програмний інтерфейс збереження даних Unity. Статистика відправляється на цей сервер, після опрацювання вона доступна для адміністратора (рис. 6).

Є низка базових метрик для опрацювання адміністратором. Метрики створюють за допомогою окремого модуля. Статистику можна зберігати в окремий файл. Доступні декілька варіантів представлення інформації.

Висновки. Під час дослідження методів синтаксичного аналізу коду виділено базові алгоритми аналізу та розглянуто найцікавіші синтаксичні аналізатори коду ANTLR, Bison, JavaCC. Розроблено власний алгоритм синтаксичного аналізу. Запропонований алгоритм дає змогу легко вносити корективи у граматику і, відповідно, в алгоритми обробки виразів. За допомогою коду лексеми забезпечується посилання на модуль обробки, а за допомогою пріоритету лексеми – порядок обробки у синтаксичному розборі. Оскільки аналіз здійснюється через розбір пари лексем, то швидкість обробки операторів вища, ніж за класичними алгоритмами через направлений перебір варіантів. При цьому дещо ускладнюється алгоритм лексичного аналізатора, що суттєво не впливає на загальну швидкість синтаксичного аналізу. Другою перевагою такого методу є простота опису правил синтаксичного розбору операторів та виразів.

Реалізований візуальний навчальний комплекс для студентів готовий до практичного використання і може бути застосований для навчання базових принципів програмування, а також для навчання мови програмування C#.

Список літератури:

1. Ахо А., Сети Р., Ульман Д. Компиляторы: принципы, технологии и инструменты. Москва: Издательский дом «Вильямс», 2008. 1184 с.
2. Bowling E. The evolution of Lotus e-Learning Software. 2009. URL: http://www.ibm.com/developerworks/lotus/library/ls-elearning_evolution
3. Хопкрофт Є. Система дистанционного обучения СДТ REDCLASS. 2009. URL: http://www.redcenter.ru/?sid=439.
4. Пратт Т., Зелковиц М. Языки программирования. Разработка и реализация. Издательство Питер, 2002. 688 с.
5. Вирт Н. Гуткнехт Ю. Разработка ОС и компилятора. Проект Оберон. Москва: ДМК Пресс, 2012. 560 с.
6. Салапатов В.І. Синтаксичний аналіз із розподілом лексем на групи. Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : збірник наукових праць. 2008. № 49. С. 30–33.

СИНТАКСИЧЕСКИЙ АНАЛИЗ КОДА ДЛЯ СИСТЕМЫ ДИСТАНЦИОННОГО ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ C#

Исследование посвящено разработке алгоритма анализатора кода для системы дистанционного обучения языкам программирования. Полученный результат заключается в модификации метода син-

таксического анализа кода на основе итеративного алгоритма. Также в работе построена модель синтаксического анализатора с распределением лексем. Данная модель является математическим обеспечением программного комплекса дистанционного обучения программированию. Кроме того, в работе изложен подход к реализации визуальных распределенных учебных комплексов с использованием COTS-архитектуры.

Ключевые слова: *синтаксический анализатор, итеративный алгоритм, анализ программного кода, дистанционное обучение, C#.*

SYNTACTIC ANALYSIS OF THE CODE
FOR THE DISTANCE LEARNING SYSTEM FOR PROGRAMMING IN LANGUAGE C#

The research is devoted to the code analyzer algorithm development for the remote learning system of the C# language. The modifying of the parsing code method based on iterative algorithm is the main result of the research. The model of the syntactic analyzer with the distribution of tokens also constructed in the work. This model is a mathematical support for the distance learning software. In addition, the article describes using COTS architecture in the visual distributed training complexes development.

Key words: *parser analyzer, iterative algorithm, program code analysis, distance learning, C#.*