ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»

ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
ФОРМА НАВЧАННЯ ДЕННА
КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА СОЦІАЛЬНОЇ ІНФОРМАТИКИ

Допускається до захисту

Завідувач кафедри _____О.О. Ємець
(підпис)

«_____»_____2020 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО БАКАЛАВСЬКОЇ РОБОТИ

на тему

ТРЕНАЖЕР З ТЕМИ «ПОБУДОВА ПРЕДИКАТИВНИХ СИНТАКСИЧНИХ АНАЛІЗАТОРІВ» ДИСТАНЦІЙНОГО НАВЧАЛЬНОГО КУРСУ «ТЕОРІЯ ПРОГРАМУВАННЯ» ТА РОЗРОБКА ЙОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

зі спеціальності 122 «Комп'ютерні науки»

Виконавець роботи Белінський Олексій Борисович
_____ «___»_____2020р.
(підпис)

Науковий керівник к.ф.-м.н., доц. Черненко Оксана Олексіївна
_____ «___»_____2020р.
(підпис)

ПОЛТАВА 2020р.

ЗМІСТ

ВСТУП

Дистанційна освіта стала рішенням для актуальних проблем навчання, вибудував власні традиції та моделі, які широко впроваджуються в усі етапи отримання знань (від шкільного та вищого, до корпоративного рівня). Широкий розвиток інформаційних та комунікаційних технологій дало такій формі «нові горизонти», дозволило втілити в життя раніше недоступні можливості.

Фактично, дистанційне навчання на практиці реалізує те, що прагне зробити традиційна школа: забезпечити рівний та всеосяжний доступ до знань, розкрити здібності та таланти, прищепити необхідні навички та якості. Якщо додати до цього можливість впровадження всіх сучасних комунікативних засобів, інноваційних методів та передових напрацювань, то ми отримуємо ту саму форму, яка може по-справжньому підготувати учня або студента, дати актуальні та сучасні знання і навички.

Метою роботи є реалізація тренажеру з теми «Побудова предикативних синтаксичних аналізаторів» дистанційного навчального курсу «Теорія програмування».

Об’єктом розробки в даній роботі є процес дистанційного навчання математичним дисциплінам.

Предметом розробки є тренажер для закріплення знань з побудови предикативних синтаксичних аналізаторів.

Головне завдання – розробити алгоритм роботи тренажеру відповідно до теми роботи та по ньому розробити програму.

Перелік використаних методів – метод побудови предикативних синтаксичних аналізаторів та програмні засоби мови Java.

При реалізації тренажеру використано середовище розробки програм NetBeans IDE та мова програмування Java.

Тренажер готовий до використання в дистанційному курсі «Теорія програмування».

Робота складається з чотирьох розділів. У першому розділі розглянуто постановку задачі тренажеру. У другому розділі описано актуальність теми роботи, огляд тренажерів. У третьому розділі представлено основні поняття про синтаксичний аналіз та предиктивний синтаксичний аналізатор, алгоритм роботи тренажеру та його блок-схему. У четвертому розділі – описано обґрунтування вибору програмних засобів, процес програмної реалізації та інструкцію програми.

Обсяг пояснювальної записки: 54 стор., в т.ч. основна частина - 45 стор., джерела - 9 назв.

1. ПОСТАНОВКА ЗАДАЧІ

Зміст роботи повинен відповідати завданню затвердженим керівником та завідувачем кафедри.

Розглянемо основні положення викладення матеріалу [1]:

- чіткість та логічна послідовність викладення матеріалу;
- переконливість аргументації;
- стислість і точність формулювань, які виключають можливість неоднозначного тлумачення; коректність викладення результатів дослідження;
- обґрунтованість рекомендацій та пропозицій.

У роботі повинні бути відображеними [1]:

- актуальність тематики та відповідність до сучасного стану науки, техніки і питань виробництва;
- обґрунтування вибраного напрямку досліджень, методів розв’язку задачі та їх порівнянь оцінки;
- аналіз та узагальнення існуючих результатів;
- розробка загальної методики проведення досліджень;
- характер і зміст виконаних теоретичних досліджень та розрахунків, методи досліджень;
- обґрунтування необхідності проведення експериментальних досліджень, принцип дії розроблених програм, характеристики цих програм, оцінка похибок розрахунків, отримані експериментальні дані; теоретичний (практичний) аналіз розроблених алгоритмів (програм);
- блок-схема програми (алгоритму);
- контрольні приклади;
- оцінка повноти розв’язку поставленої задачі;

- наукова або практична цінність виконаної роботи, виклад наукової новизни, якщо вона є.

Для дистанційного курсу «Теорія програмування» потрібно розробити програму-тренажер.

Тренажер повинен містити:

- Головне вікно;
- Ознайомчий матеріал за темою;
- Тести;
- Довідкову інформацію;
- Повідомлення про завершення.

При неправильній відповіді слід реалізувати виведення довідкової інформації.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Актуальність теми роботи

Відразу необхідно зазначити, що не існує двох однакових програм для дистанційного навчання. Можуть бути подібні методи та форми, проте, в цілому навчання завжди буде відрізнятися. Це властиво будь-якій загальноосвітній чи вищій школі, де навіть при наявності єдиних стандартів та загальних вимог, фактичне навчання трохи відрізняється. Ці відмінності випливають з природних потреб навчальних закладів, де фактично працюють різні викладачі, діють різні академічні школи, встановлюється різний акцент на напрямку навчання.

Дистанційна форма пішла ще далі, навчання кожного учня, в цій системі, це індивідуальний та гнучкий процес. І хоча всі випускники відповідають формальним вимогам, існує можливість вибудувати власний навчальний план, освоювати матеріал у своєму темпі, робити акцент на тих сферах, котрі цікаві учневі. Саме така гнучкість та адаптивність дає величезну перевагу дистанційному навчанню, вона дозволяє не тільки оптимізувати час та програму, а й знизити стрес-фактор для всіх учасників процесу.

Більш постійна взаємодія та зворотний зв'язок від викладача значно підвищує фактичний контроль за освоєнням матеріалу, дає додаткову можливість допомогти учневі, дозволяє вивчити тему, що цікавить глибше.

З якими ж проблемами допомагає впоратися дистанційне навчання?

1. Проблема відстані та великих міст.

І хоча практично в будь-якому місті є свої школи і ВНЗ, проте, найвідоміші заклади, які пропонують дійсно якісну освіту, розташовані в обласних центрах або столиці. Крім того, багато університетів, котрі зацікавлять абітурієнта, знаходяться закордоном. В цьому випадку, дистанційна форма навчання - чудова можливість отримати знання,

навички та диплом. Немає потреби переїжджати на постійне місце проживання, значно спрощена і бюрократична процедура вступу до іноземного навчального закладу.

З іншого боку, така форма навчання пропонує куди більше, ніж традиційна заочна форма. Крім завдань та навчальних матеріалів, дистанційне навчання передбачає розширення аудиторних рамок для учнів і викладачів, інтенсивну взаємодію і ефективну форму контролю.

2. Часові обмеження

Це проблема більш актуальна для тих, хто поєднує роботу та навчання. Іншими словами, для тих, хто не може відвідувати очні заняття, знаходиться та працює в іншому місті, але хоче отримати більше ніж проста заочна форма - це ідеальний варіант.

Він передбачає не тільки гнучку програму, але і можливість працювати у своєму ритмі, виділити кілька годин на день на опрацювання і навчання коли це буде можливо та зручно: в перерві, після роботи, у вихідний та ін.

3. Питання ціни

Дистанційна освіта нерідко дешевше ніж очне відділення. Платне навчання на очному відділенні в престижній приватній школі, навіть в Україні, не всім по кишені. З іншого боку, адміністрації немає необхідності витрачати кошти на оренду та утримання навчальних приміщень, що неминуче скорочує витрати. Все це дозволяє запропонувати більш зручні умови та доступну ціну навчання.

4. Питання якості

Існує багато шкіл та інститутів, котрі славляться своїми досягненнями в певній галузі. Нерідко, така спеціалізація навчального закладу приносить йому значну частку учнів та студентів. Дистанційна форма дозволяє працювати з кращими викладачами, за найдосконалішою програмою, досягати високих результатів навіть не перебуваючи на території навчального закладу.

Звичайно, в даному питанні багато залежить від самого учня та студента, від його вміння організувати себе та освоїти матеріал. Однак, для дистанційного навчання передбачені форми контролю аналогічні очній формі, можливе відвідування певних занять або сесії особисто [2-3].

2.2. Огляд тренажерів

Було розглянуто декілька тренажерів:

- тренажер «Синтаксичний аналіз» (дисципліна «Теорія програмування»);
- тренажер з теми «Рекурсивні функції» дистанційного навчального курсу «Теорія алгоритмів»;
- тренажер з теми «Нормальні алгоритми» з дисципліни «Теорія алгоритмів»

В тренажері «Синтаксичний аналіз» на головній сторінці виводиться інформація про нього, умова прикладу і завдання з варіантами відповіді (рис. 2.1).

Якщо вказати правильну відповідь, то заповнюються таблиці і відбувається перехід до наступного кроку (рис. 2.2).

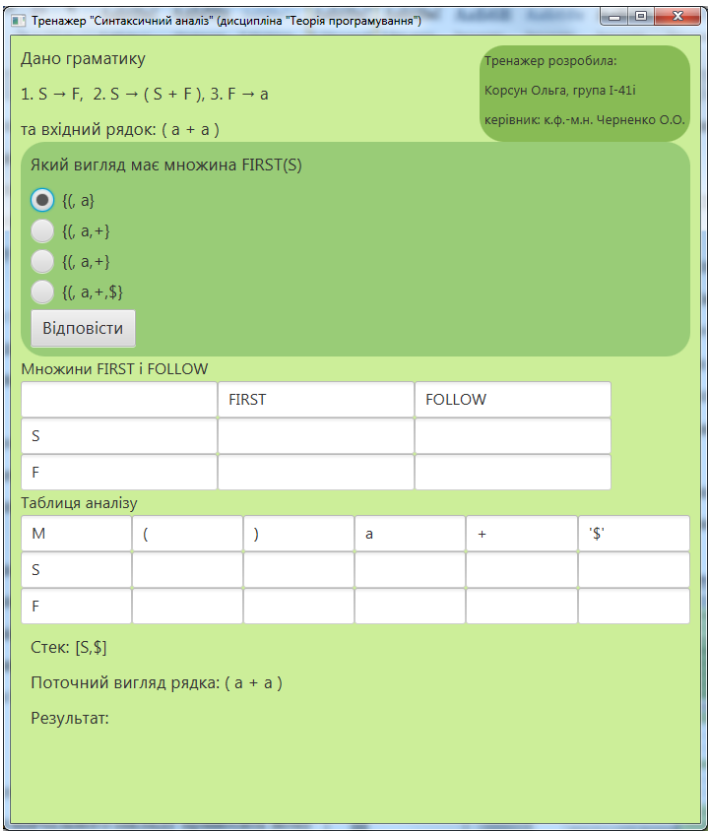

Рисунок 2.1 – Тренажер «Синтаксичний аналіз»

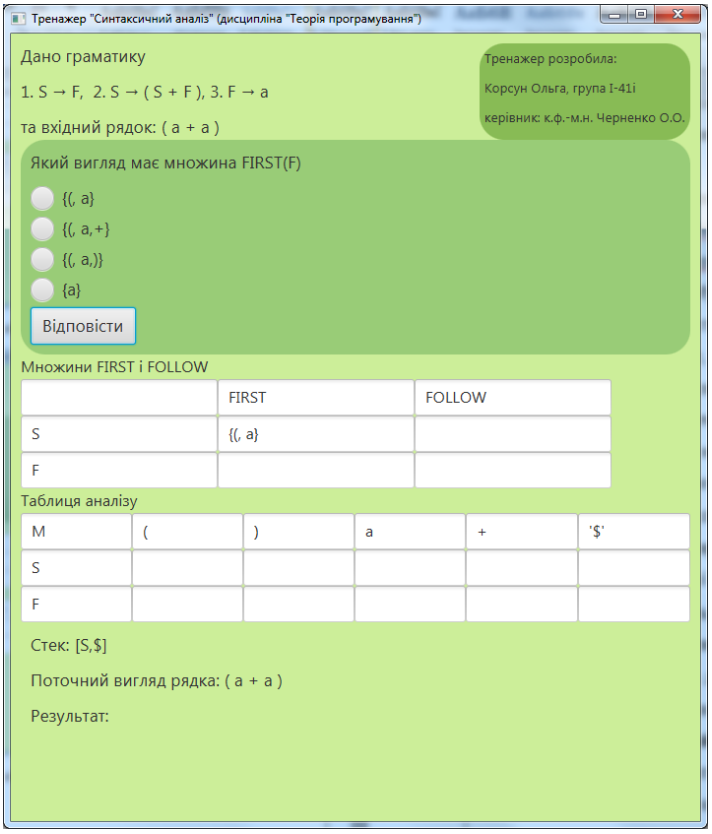

Рисунок 2.2 – Тренажер «Синтаксичний аналіз»

Якщо один раз відповісти неправильно, то з’явиться повідомлення про помилку (рис. 2.3).

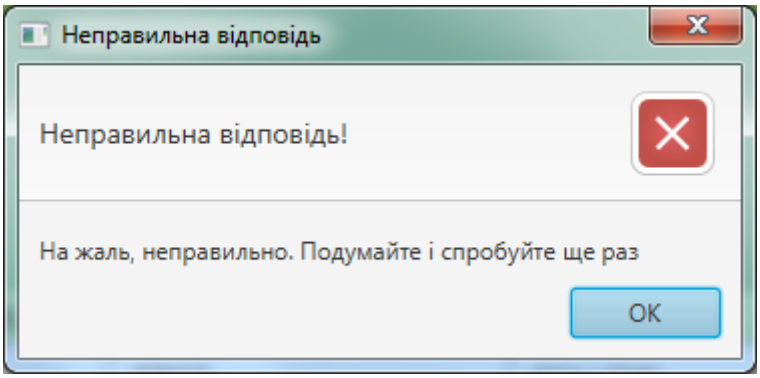

Рисунок 2.3 – Повідомлення про помилку

Повторивши помилку, з’явиться повідомлення з підказкою (рис. 2.4).

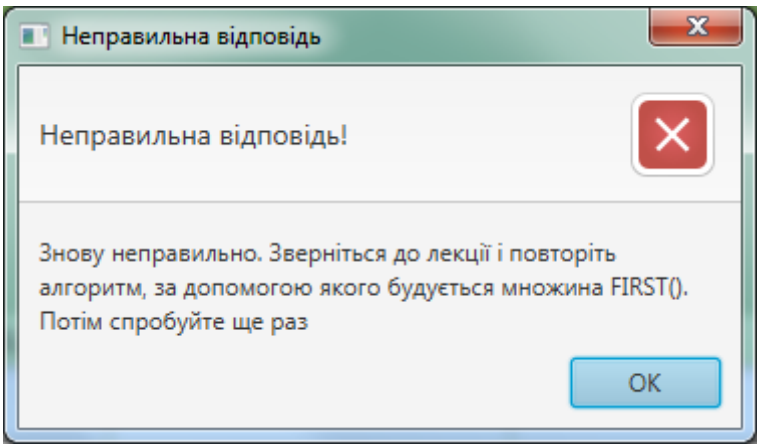

Рисунок 2.4 – Повідомлення з підказкою

На останньому кроці виводиться результат (рис. 2.5).

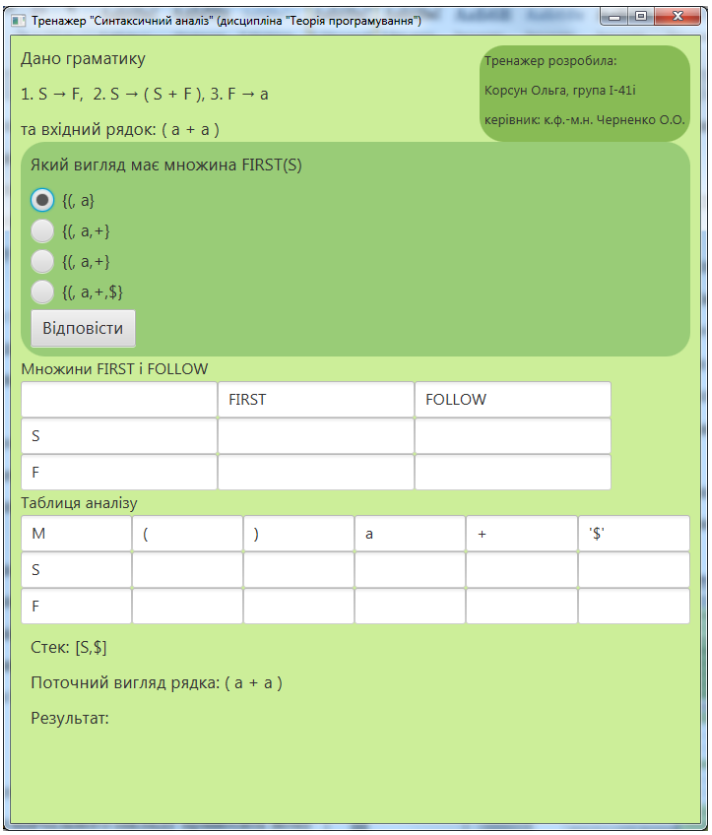

Рисунок 2.5 – Результат роботи тренажеру

Тренажер з теми «Рекурсивні функції» пропонує переглянути теоретичний матеріал або перейти до тренінгу (рис. 2.6).

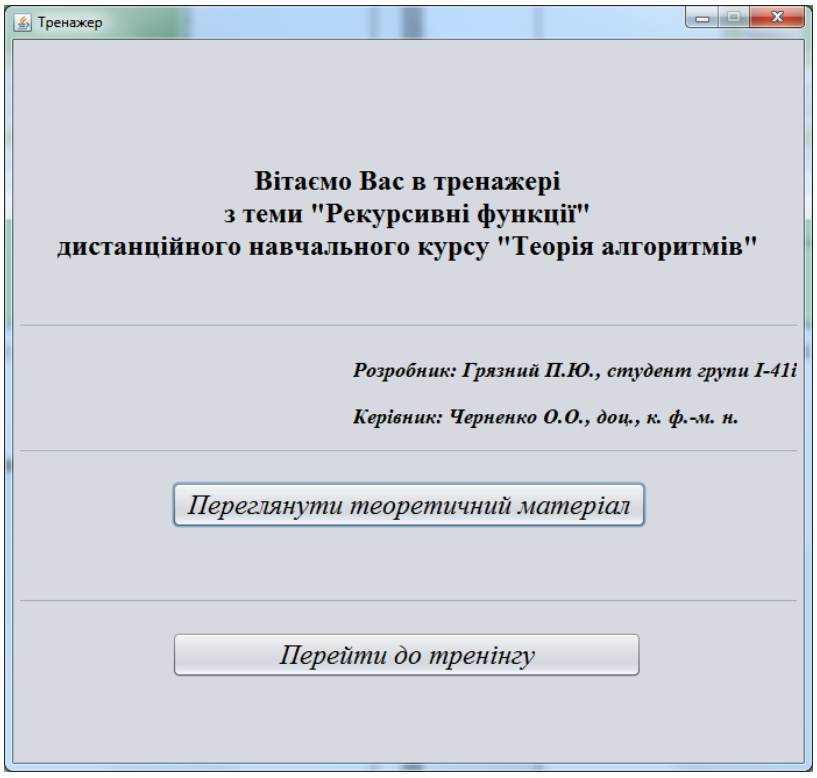

Рисунок 2.6 – Тренажер з теми «Рекурсивні функції»

Спочатку йдуть теоретичні питання, а потім практичні.

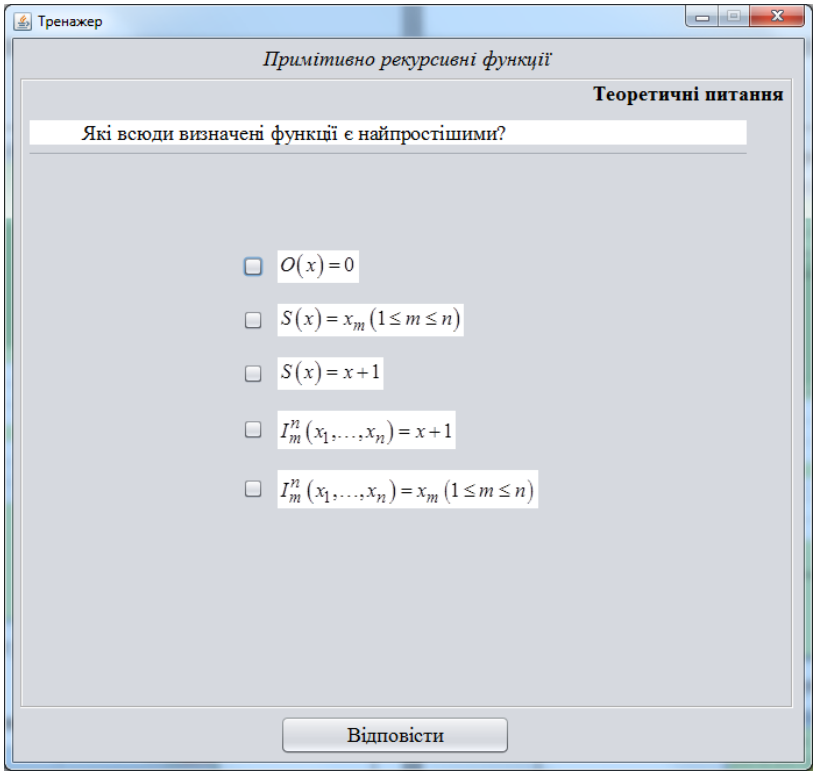

Рисунок 2.7 – Теоретичні питання

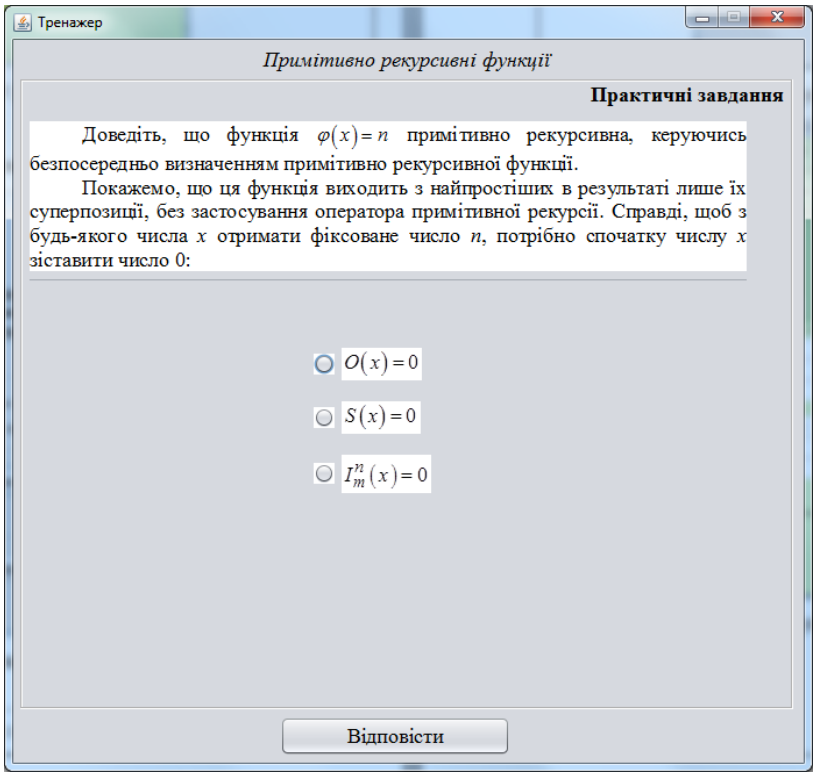

Рисунок 2.8 – Практичні питання

Якщо допущено помилку, то виведеться повідомлення про це (рис. 2.9).

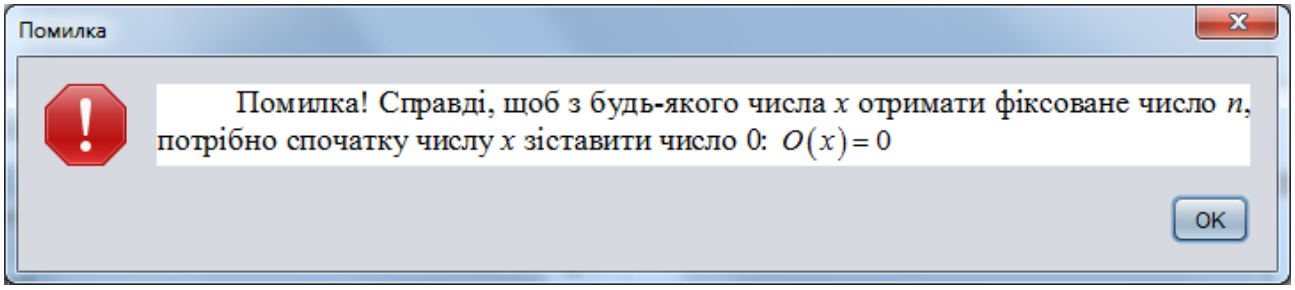

Рисунок 2.9 – Повідомлення про помилку

В кінці пропонується завершити роботу або розпочати спочатку (рис. 2.10).

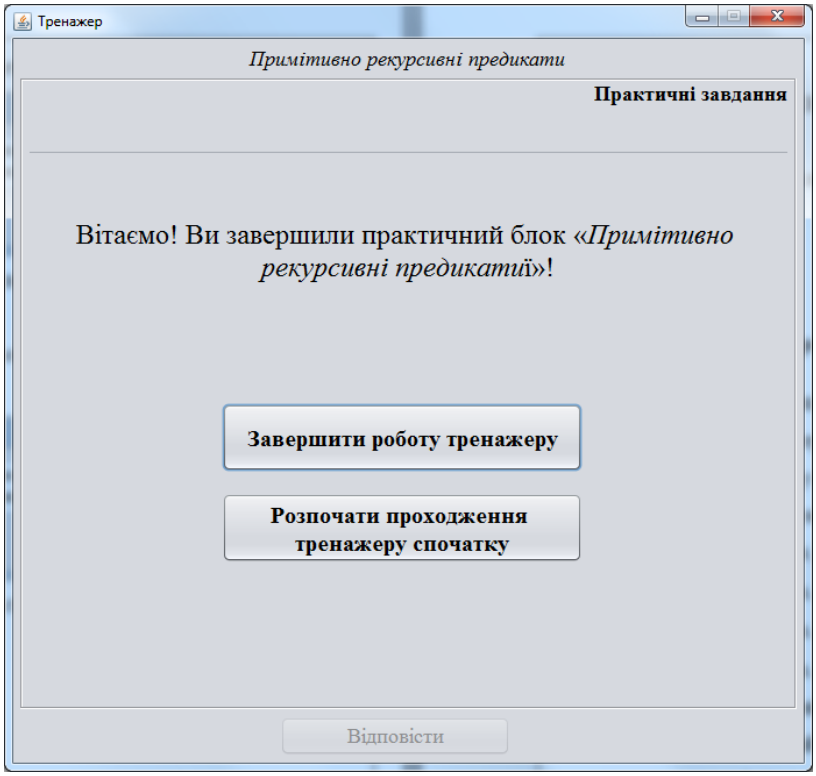

Рисунок 2.10 – Кінцева сторінка тренажеру

Тренажер з теми «Нормальні алгоритми» має схожу головну сторінку (рис. 2.11). Він має схожість з вже розглянутими.

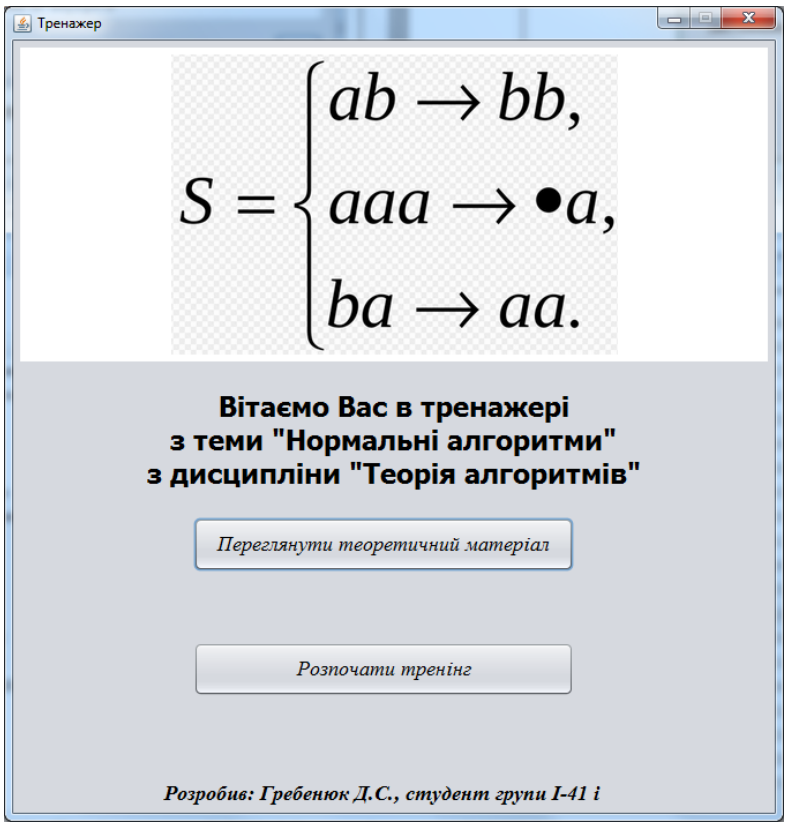

Рисунок 2.11 – Тренажер з теми «Нормальні алгоритми»

В ньому спочатку виводиться умова задачі. Далі покроково її необхідно розв’язати (рис. 2.12-2.13).

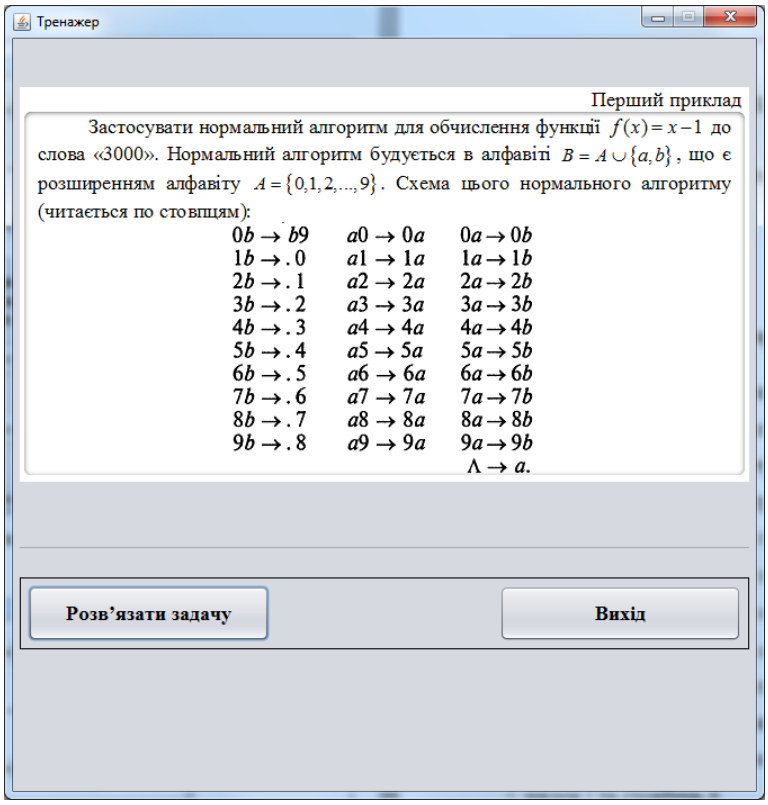

Рисунок 2.12 – Умова задачі

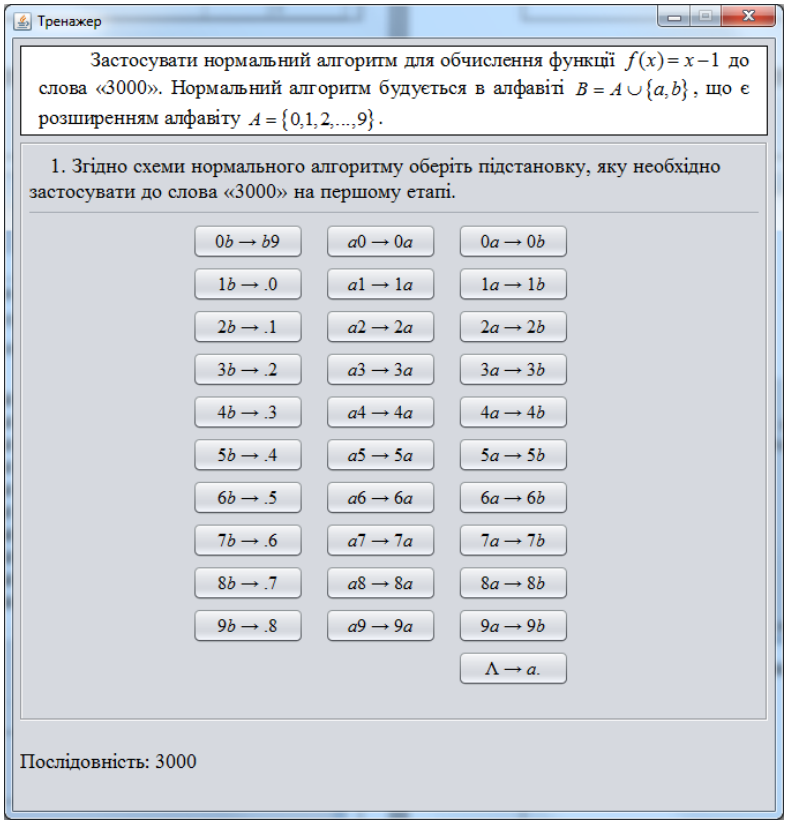

Рисунок 2.13 – Покрокове розв’язання

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Синтаксичний аналіз та предиктивний синтаксичний аналізатор

Предиктивний синтаксичний аналізатор – це синтаксичний аналізатор, що працює методом рекурсивного спуску без повернень (відкотів). Це можливо, якщо з граматики видалена ліва рекурсія і вона лівофакторизована.

У багатьох випадках акуратне розроблення граматики, видалення з неї лівої рекурсії та її ліва факторизація дозволяють одержати граматику, яка може бути проаналізована синтаксичним аналізатором, що використовує метод рекурсивного спуску і не потребує відкоту (предиктивним аналізатором).

Нерекурсивний предиктивний аналізатор можна побудувати за допомогою явного використання стека замість неявного при рекурсивних викликах. Ключова проблема предиктивного аналізу полягає у визначенні продукції, яку потрібно застосувати до нетермінала. Для пошуку продукції може бути використана таблиця розбору.

Модель предиктивного синтаксичного аналізатора, що керується таблицею, показана на рис. 3.1.

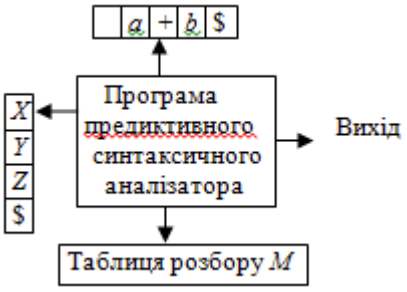

Рисунок 3.1 - Схема предиктивного аналізатора

Аналізатор має вхідний буфер, стек, таблицю розбору і вихідний потік. Буфер містить вхідний рядок, за яким йде правий кінцевий маркер $ – ознака кінця рядка. Стек містить послідовність символів граматики з

$ на дні. Спочатку стек містить початковий символ граматики безпосередньо над символом $. Таблиця розбору – це двовимірний масив *M(A,a)*, де *A* – нетермінал; *a* – термінал або символ $.

Аналізатор керується програмою, що працює у такий спосіб. Програма розглядає *X* – символ на вершині стека і *a* – поточний вхідний символ. Ці два символи визначають дію аналізатора. Є три можливості.

1. Якщо *X=a=$*, аналізатор зупиняється і повідомляє про успішне закінчення розбору.
2. Якщо *X=a≠$*, аналізатор видаляє *X* зі стека і просуває покажчик вхідного потоку на наступний символ.
3. Якщо *X* – нетермінал, програма розглядає запис *M(X,a)* з таблиці розбору *M*. Цей запис являє собою або *X*-продукцію граматики, або запис про помилку. Якщо, наприклад, *M(X,a)={X→UVW}*, аналізатор заміняє *X* на вершині стека на *WVU* (з *U* на вершині). Будемо вважати, що аналізатор на виході просто друкує використану продукцію. Якщо *M(X,a)=error*, аналізатор викликає підпрограму аналізу помилок.

Алгоритм 1. Нерекурсивний предикативний аналіз

Спочатку аналізатор знаходиться у конфігурації з $*S* (на вершині – стартовий символ *S* граматики *G*) і рядком *w*$ у вхідному буфері.

Установити покажчик *iP* на перший символ *w*$;

repeat

Позначимо через *X* символ на вершині стека, а через *a* – символ, на який вказує *iP*.

if *X* – термінал або $ **then**

if *X=a* **then** видалити *X* зі стека і перемістити *iP*

else *error*

else if $M(X,a)=X\rightarrow Y_1Y_2\ldots Y_k$ **then** /**X* – нетермінал*/

begin

видалити X зі стека;

помістити у стек $Y_k, Y_{k-1}, \ldots, Y_1$ на вершині стека;

вивести продукцію $X \rightarrow Y_1 Y_2 \ldots Y_k$

end

else *error* /*вхід таблиці *M* порожній*/

until $X=\$$. /*стек порожній*/

Приклад 2.Розглянемо граматику арифметичних виразів:

$E \rightarrow TE', E' \rightarrow +TE' \mid \varepsilon,$

$T \rightarrow FT', T' \rightarrow *FT' \mid \varepsilon,$

$F \rightarrow (E) \mid id$

Для вхідного потоку **id+id*id** предиктивний синтаксичний аналізатор робить послідовність кроків, зображену у табл. 3.1. Покажчик входу вказує на самий лівий символ у колонці “Вхід”. Якщо уважно проаналізувати дії аналізатора, то видно, що його вихід збігається з послідовністю продукцій, які використовуються в лівому породженні. Якщо до прочитаних вхідних символів приписати символи у стеку (від вершини до дна), то одержимо лівосентенціальну форму в породженні.

Таблиця 3.1 – Синтаксичний аналіз вхідного потоку **id+id*id**

Стек	Вхід	Вихід
$E	id+id*id$	E→TE'
$E' T	id+id*id$	T→FT'
$E' T' F	id+id*id$	F→id
$E' T' id	id+id*id$	T'→ε
$E' T'	+id*id$	E'→+TE'
$E'	+id*id$	
$E' T+	+id*id$	T→FT'
$E' T	id*id$	F→id
$E' T' F	id*id$	T'→*FT'
$E' T' id	id*id$	F→id
$E' T'	*id$	T'→ε
$E' T' F*	*id$	E'→ε
$E' T' F	id$	
$E' T' id	id$	
$E' T'	$	
$E'	$	
$	$	

При побудові предиктивного аналізатора корисними виявляються дві функції, зв'язані з граматикою G. Ці функції, FIRST і FOLLOW, дозволяють побудувати таблицю предиктивного розбору для G, якщо, звичайно, це можливо. Множини, що породжуються цими функціями, можуть, крім того, бути використані при відновленні після помилок.

Нехай a – довільний рядок символів граматики. Функція FIRST(a) - це множина терміналів, з яких починаються рядки, виведені з a. Якщо $a \overset{*}{\Rightarrow} \varepsilon$, то ε також належить FIRST(a). Для побудови FIRST(X) для всіх символів X граматики застосуємо такий алгоритм.

Алгоритм 2. Побудова множини FIRST для символів граматики.

Крок 1 Якщо X – термінал, то FIRST(X) = {X}; якщо X - нетермінал, то FIRST(X) = {}.

Крок 2 Якщо є продукція $X \rightarrow \varepsilon$, то додати ε до FIRST($X$).

Крок 3. Якщо X – нетермінал і є продукція $X \to Y_1 Y_2 \ldots Y_k$, то включити a у FIRST(X), якщо для деякого a FIRST(Y_i) та ε належить усім множинам FIRST(Y_1),…, FIRST(Y_{i-1}), тобто $Y_1 \ldots Y_{i-1} \Rightarrow \varepsilon$. Якщо ε належить FIRST(Y_i) для всіх $i = 1, 2, \ldots, k$, то додати ε до FIRST(X).

Наприклад, усе, що належить FIRST(Y_1) належить також і FIRST(X). Якщо з Y_1 не виводиться ε, то ми нічого більше не додаємо до FIRST(X), але якщо $Y_1 \Rightarrow \varepsilon$, то додаємо FIRST($Y_2$) і т.д.

Тепер FIRST для будь-якого рядка $X_1 X_2 \ldots X_n$ можна обчислити у такий спосіб. Вважаємо FIRST($X_1 X_2 \ldots X_n$) = {}. Додамо до FIRST($X_1 X_2 \ldots X_n$) усі не ε-символи FIRST(X_1). Додамо також всі не ε-символи з FIRST(X_2), якщо $\varepsilon \in$ FIRST(X_1), всі не ε-символи з FIRST(X_3), якщо ε належить як FIRST(X_1), так і FIRST(X_2), і т.д. Нарешті, додамо ε до FIRST($X_1 X_2 \ldots X_n$), якщо для всіх і FIRST(X_i) містить ε.

Функція FOLLOW(A) для нетермінала A - це множина терміналів a, що можуть з'явитися безпосередньо праворуч від A у деякій сентенціальній формі, тобто множина терміналів a таких, що існує породження вигляду $S\alpha \Rightarrow Aa\beta$ для деяких α і β. Відзначимо, що між A і a у процесі виведення можуть з'явитися нетермінальні символи, з яких виводиться ε. Якщо A може виявитися крайнім правим символом деякої сентенціальної форми, то \$ належить FOLLOW(A).

Для обчислення FOLLOW(A) для нетермінала A застосуємо такий алгоритм.

Алгоритм 3. Побудова FOLLOW(X) для всіх X -нетерміналів граматики.

Крок 1 Помістити \$ у FOLLOW(S) , де S – початковий символ і \$ – правий кінцевий маркер.

Крок 2 Якщо є продукція $A \to \alpha B \beta$, то усі елементи з множини FIRST(β), за винятком ε , додати до FOLLOW(B).

Крок 3 Якщо є продукція $A \to \alpha B$ або $A \to \alpha B\beta$, де FIRST(β) містить ε (тобто $\beta \overset{*}{\Rightarrow} \varepsilon$), то усі елементи з множини FOLLOW(A) додати до FOLLOW(B).

Приклад 3.Запишемо функції FIRST і FOLLOW для граматики з прикладу 2.

$E \to TE'$,

$T \to FT'$,

$F \to (E) \mid id$

Для неї:

FIRST(E) = FIRST(T) = FIRST(F) = {(,id};

FIRST(E') = {+, ε};

FIRST(T') ={*, ε};

FOLLOW(E) = FOLLOW(E') = {),$};

FOLLOW(T) = FOLLOW(T') = {+,),$};

FOLLOW(F) = {+,*,),$}.

Наприклад, **id** і ліва дужка додаються до FIRST(F) на кроці 3 алгоритму для FIRST, оскільки FIRST(**id**) = {**id**} і FIRST('(') = {**(**} у відповідності з кроком 1. На кроці 3 при відповідно до продукції $T \to FT'$ до FIRST(T) додаються також **id** і ліва дужка. На кроці 2 включається ε.

На кроці 1 обчислення множини FOLLOW у FOLLOW(E) включаємо $. На кроці 2, використовуючи продукцію $F \to (E)$, до FOLLOW(E) додається також права дужка. На кроці 3, застосованому до правила $E \to TE'$, включаються $ і права дужка. Оскільки вони також потрапляють у FOLLOW(T). Відповідно до продукції $E \to TE'$ на кроці 2 у FOLLOW(T) включається усе з , відмінне від ε.

Для конструювання таблиць предиктивного аналізу даної граматики G може бути використаний алгоритм, що ґрунтується на такій ідеї. Припустимо, що $A \to \alpha$ – продукція граматики і $a \in$ FIRST(α). Тоді

аналізатор робить розгортання A за α, якщо вхідним символом є a. Труднощі виникають, коли $\alpha = \varepsilon$ або $\alpha \Rightarrow \varepsilon$. У цьому випадку потрібно розгорнути A у α , якщо поточний вхідний символ належить FOLLOW(A), або якщо з вхідного потоку одержано \$ і \$ ∈ FOLLOW(A).

Алгоритм 4. Побудова таблиць предиктивного аналізу

Для кожної продукції $A \to \alpha$ граматики виконати кроки 1 і 2.

Крок 1 Для кожного термінала a з FIRST(α) додати $A \to \alpha$ в *M(A,a)*.

Крок 2 Якщо $\varepsilon \in$ FIRST(α), додати $A \to \alpha$ в *M(A,b)* для кожного термінала b з FOLLOW(A). Якщо $\varepsilon \in$ FIRST(α) і \$ ∈ FOLLOW(A), додати $A \to \alpha$ в *M(A,\$)*.

Крок 3 Прийняти, що всі невизначені входи таблиці M вказують на помилку.

Приклад 4. Застосуємо алгоритм 4 до граматики з прикладу 2. Оскільки FIRST(T*E'*) = FIRST(*T*) = {(,id}, у відповідності з продукцією $E \to TE'$ входи *M(E,(* та *M(E,*id*)* стають рівними $E \to TE'$ [4-6].

Таблиця 3.2 - Предиктивний аналізатор для граматики з прикладу 2

Нетермінал	Вхідний символ					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to T\ E'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to F\ T'$		
T'		$T' \to \varepsilon$	$T' \to *F\ T'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to$ **id**			$F \to (E)$		

3.2. Розробка алгоритму роботи тренажеру

На головній сторінці відображається тема тренажеру, інформація про розробника і керівника, пропонується відкрити теоретичний матеріал або перейти до виконання тестів.

Окремими сторінками розміщується довідкова інформація з основних завдань:

- робота програми аналізатора;
- алгоритм 1. Нерекурсивний предиктивний аналіз;
- алгоритм 2. Побудова множини FIRST для символів граматики;
- алгоритм 3. Побудова FOLLOW(X) для всіх X-нетерміналів граматики;
- алгоритм 4. Побудова таблиць предикативного аналізу.

Крок 1. Завдання: Аналізатор керується програмою, що працює у такий спосіб. Програма розглядає X – символ на вершині стека і a – поточний вхідний символ. Ці два символи визначають дію аналізатора. Є три можливості.

1. Якщо $X=a=\$$,

Варіанти відповіді:

- аналізатор зупиняється і повідомляє про успішне закінчення розбору.
- аналізатор видаляє X зі стека і просуває покажчик вхідного потоку на наступний символ.
- програма розглядає запис $M(X,a)$ з таблиці розбору M. Цей запис являє собою або X-продукцію граматики, або запис про помилку. Якщо $M(X,a)=error$, аналізатор викликає підпрограму аналізу помилок.

Якщо відповідь – перший варіант, то перехід на наступний крок. Якщо ні – відбувається перехід до довідкової інформації.

Крок 2. Завдання: Аналізатор керується програмою, що працює у такий спосіб. Програма розглядає X – символ на вершині стека і a – поточний вхідний символ. Ці два символи визначають дію аналізатора. Є три можливості.

2. Якщо $X=a\neq\$$,

Варіанти відповіді:

- аналізатор зупиняється і повідомляє про успішне закінчення розбору.
- аналізатор видаляє X зі стека і просуває покажчик вхідного потоку на наступний символ.
- програма розглядає запис $M(X,a)$ з таблиці розбору M. Цей запис являє собою або X-продукцію граматики, або запис про помилку. Якщо $M(X,a)=error$, аналізатор викликає підпрограму аналізу помилок.

Якщо відповідь – другий варіант, то перехід на наступний крок. Якщо ні – відбувається перехід до довідкової інформації.

Крок 3. Завдання: Аналізатор керується програмою, що працює у такий спосіб. Програма розглядає X – символ на вершині стека і a – поточний вхідний символ. Ці два символи визначають дію аналізатора. Є три можливості.

3. Якщо X – нетермінал,

Варіанти відповіді:

- аналізатор зупиняється і повідомляє про успішне закінчення розбору.
- аналізатор видаляє X зі стека і просуває покажчик вхідного потоку на наступний символ.
- програма розглядає запис $M(X,a)$ з таблиці розбору M. Цей запис являє собою або X-продукцію граматики, або запис про

помилку. Якщо $M(X,a)=error$, аналізатор викликає підпрограму аналізу помилок.

Якщо відповідь – третій варіант, то перехід на наступний крок. Якщо ні – відбувається перехід до довідкової інформації.

Крок 4. Завдання: Нерекурсивний предикативний аналіз.

Спочатку аналізатор знаходиться у конфігурації з $\$S$ (на вершині – стартовий символ S граматики G) і рядком $w\$$ у вхідному буфері.

Встановити послідовність кроків.

Кроки:

(2) **repeat**

(1) Установити покажчик iP на перший символ $w\$$;

(3) Позначимо через X символ на вершині стека, а через a – символ, на який вказує iP.

(14) **until** $X=\$$.

(4) **if** X – термінал або \$ **then**

(10) помістити у стек $Y_k,Y_{k-1},\ldots,Y_1$ на вершині стека;

(6) або (13) **else** *error*

(5) **if** $X=a$ **then** видалити X зі стека і перемістити iP

(7) **else if** $M(X,a)=X\rightarrow Y_1Y_2\ldots Y_k$ **then**

(8) **begin**

(9) видалити X зі стека;

(11) вивести продукцію $X\rightarrow Y_1Y_2\ldots Y_k$

(12) **end**

(6) або (13) **else** *error*

Якщо відповідь – правильно вказані номера кроків, то перехід на наступний крок. Якщо ні – відбувається перехід до довідкової інформації.

Крок 5. Завдання: Побудова множини FIRST для символів граматики.

Нехай a – довільний рядок символів граматики. Функція FIRST(a) - це множина терміналів, з яких починаються рядки, виведені з a. Якщо

$a \overset{*}{\Rightarrow} \varepsilon$, то ε також належить FIRST(a). Для побудови FIRST(X) для всіх символів X граматики застосуємо такий алгоритм.

Встановити послідовність кроків.

Кроки:

(2) Якщо є продукція $X \rightarrow \varepsilon$, то додати ε до FIRST($X$).

(3) Якщо X – нетермінал і є продукція $X \rightarrow Y_1 Y_2 \ldots Y_k$, то включити a у FIRST(X), якщо для деякого a FIRST(Y_i) та ε належить усім множинам FIRST(Y_1),…, FIRST(Y_{i-1}), тобто $Y_1 \ldots Y_{i-1} \Rightarrow \varepsilon$. Якщо ε належить FIRST(Y_i) для всіх $i = 1, 2, \ldots, k$, то додати ε до FIRST(X).

(1) Якщо X – термінал, то FIRST(X) = {X}; якщо X - нетермінал, то FIRST(X) = {}.

Якщо відповідь – правильно вказані номера кроків, то перехід на наступний крок. Якщо ні – відбувається перехід до довідкової інформації.

Крок 6. Завдання: Побудова FOLLOW(X) для всіх X-нетерміналів граматики.

Функція FOLLOW(A) для нетермінала A - це множина терміналів a, що можуть з'явитися безпосередньо праворуч від A у деякій сентенціальній формі, тобто множина терміналів a таких, що існує породження вигляду $S\alpha \Rightarrow Aa\beta$ для деяких α і β. Відзначимо, що між A і a у процесі виведення можуть з'явитися нетермінальні символи, з яких виводиться ε. Якщо A може виявитися крайнім правим символом деякої сентенціальної форми, то \$ належить FOLLOW(A).

Для обчислення FOLLOW(A) для нетермінала A застосуємо такий алгоритм.

Встановити послідовність кроків.

Кроки:

(3) Якщо є продукція $A \to \alpha B$ або $A \to \alpha B \beta$, де FIRST(β) містить ε (тобто $\beta \overset{*}{\Rightarrow} \varepsilon$), то усі елементи з множини FOLLOW(A) додати до FOLLOW(B).

(1) Помістити \$ у FOLLOW(S) , де S – початковий символ і \$ – правий кінцевий маркер.

(2) Якщо є продукція $A \to \alpha B \beta$, то усі елементи з множини FIRST(β), за винятком ε , додати до FOLLOW(B).

Якщо відповідь – правильно вказані номера кроків, то перехід на наступний крок. Якщо ні – відбувається перехід до довідкової інформації.

Крок 7. Завдання: Побудова таблиць предикативного аналізу.

Для конструювання таблиць предикативного аналізу даної граматики G може бути використаний алгоритм, що ґрунтується на такій ідеї. Припустимо, що $A \to \alpha$ – продукція граматики і $a \in$ FIRST(α). Тоді аналізатор робить розгортання A за α, якщо вхідним символом є a. Труднощі виникають, коли $\alpha = \varepsilon$ або $\alpha \Rightarrow \varepsilon$. У цьому випадку потрібно розгорнути A у α , якщо поточний вхідний символ належить FOLLOW(A), або якщо з вхідного потоку одержано \$ і \$ $\in$ FOLLOW(A).

Встановити послідовність кроків.

Кроки:

(2) Для кожного термінала a з FIRST(α) додати $A \to \alpha$ в *M(A,a)*.

(3) Якщо $\varepsilon \in$ FIRST(α), додати $A \to \alpha$ в *M(A,b)* для кожного термінала b з FOLLOW(A). Якщо $\varepsilon \in$ FIRST(α) і \$ $\in$ FOLLOW(A), додати $A \to \alpha$ в *M(A,\$)*.

(1) Для кожної продукції $A \to \alpha$ граматики виконати кроки 1 і 2.

(4) Прийняти, що всі невизначені входи таблиці M вказують на помилку.

Якщо відповідь – правильно вказані номера кроків, то перехід на наступний крок. Якщо ні – відбувається перехід до довідкової інформації.

Після проходження виводиться повідомлення про завершення, відкривається стартова сторінка.

3.3. Розробка блок-схеми, яка підлягає програмуванню

На рисунку 3.2 зображено блок-схему алгоритму роботи тренажеру.

На рисунку 3.3 зображено блок-схему алгоритму проходження тестів.

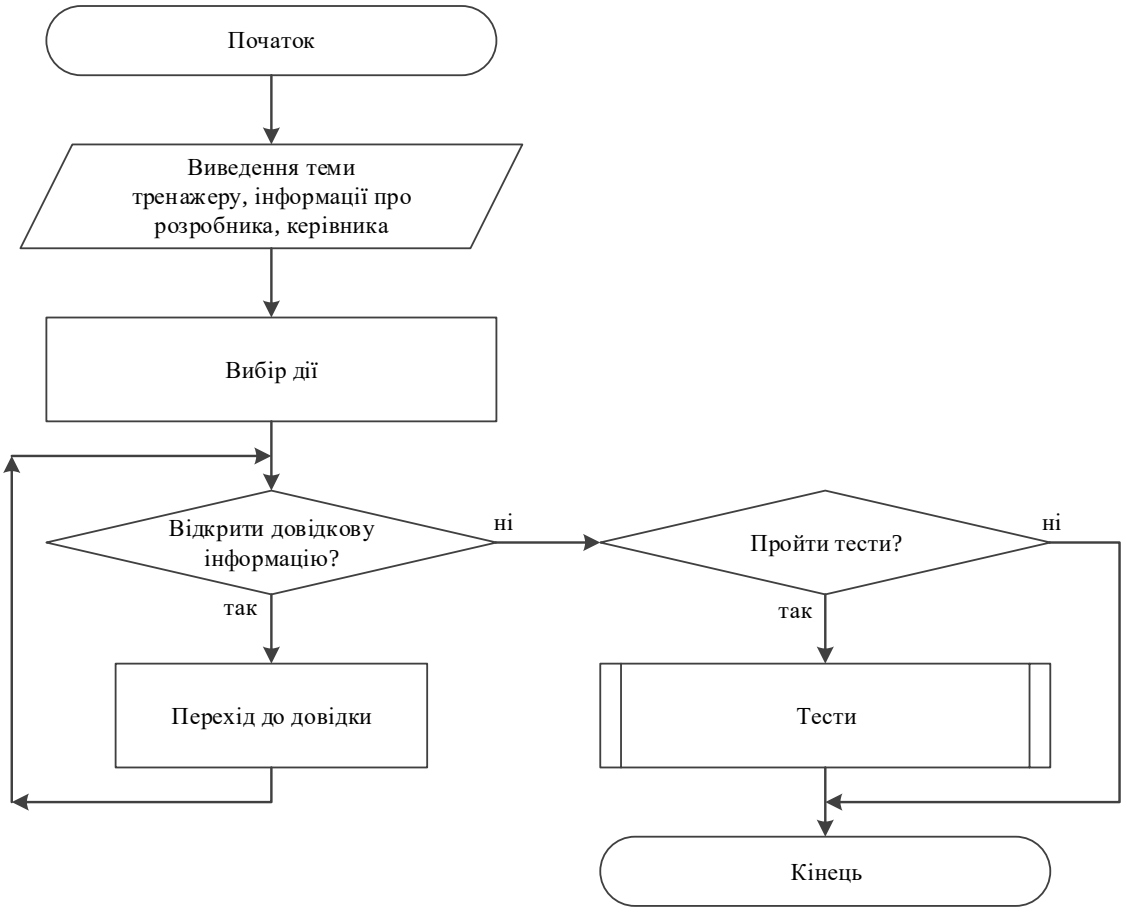

Рисунок 3.2 – Блок-схема алгоритму роботи тренажеру

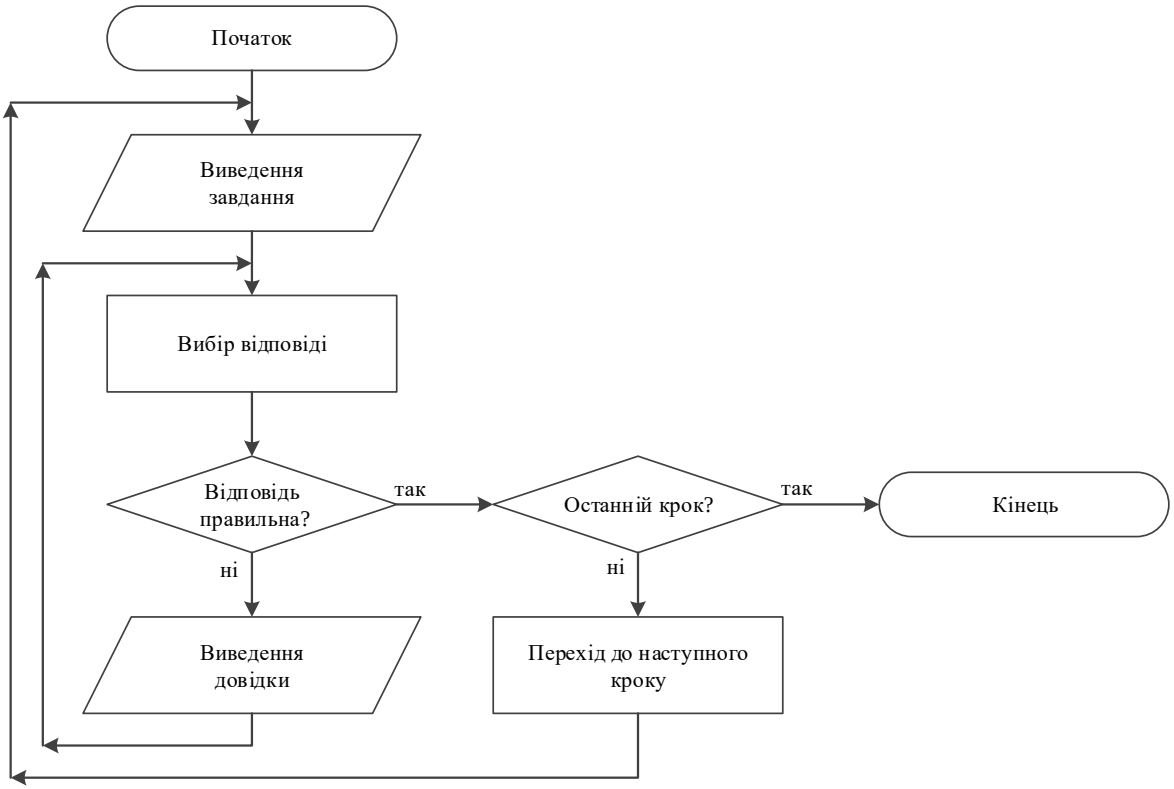

Рисунок 3.3 – Блок-схема алгоритму проходження тестів

4. ПРАКТИЧНА ЧАСТИНА

4.1. Обґрунтування вибору програмних засобів для реалізації завдання роботи

Якщо ви коли-небудь бачили успішні, живі сайти з яскравою анімацією, інтерактивними іграми, чудовими банерами, 3-D об'єктами і зображеннями - все це можливості Java. Живучи в сучасному суспільстві, ви напевно стикалися з великою різноманітністю мов програмування і платформ для розробки різних технологічних пристосувань, принаймні один раз. І ви, звичайно, здогадуєтесь, що робота з такими технологіями вимагає справжніх професіоналів, що володіють знаннями, здібностями і досвідом. Зараз вам не потрібно їх шукати - ми тут. Розробники Java у нашій компанії - це команда висококваліфікованих фахівців, які знають правильний підхід і спосіб використання та впровадження найкращих технологій цієї галузі.

Результатом роботи Java може бути створення цікавої гри або корисних бізнес-додатків, які будуть працювати за допомогою комп'ютера, але також можуть бути адаптивним для роботи на мобільних пристроях. Комплект розробки Java для запуску в компанії ArmedSoft - це інструмент Java, Java Compiler, Javadoc, Jar і Java налагоджувач (дебаггер) [7].

Причина #1. Java ще не скоро втратить актуальність

Від моменту появи 1995 року, від початку глобальної комп’ютеризації і до сьогодні — мова програмування Java стабільно користується попитом на ринку. З того часу Java довела свою затребуваність. Java зазирнула у майбутнє!

Java run everywhere! Java зробили кросплатформенною, з відкритим кодом, безкоштовною. Знову ж таки на той час це був прогрес і далекоглядне рішення, щоб зробити Java — універсальною мовою

програмуваня. Сьогодні з її допомогою можна створювати програмне забезпечення як для комп'ютерів, так і для мобільних пристроїв, але про це поговоримо трішки нище…

Віртуальна Java (JVM) популярна й для інших мов і платформ. Виявляється, більше 20 років розвитку на чолі з геніальними корпораціями привели до створення налагодженої моделі. Так що зараз існують такі мови програмування, які Scala, Groovy і Jruby, котрі компілюються з байт-кодом JVM. І знання Java допоможе Вам вивчити ці мови, оскільки в них будуть часто використовуватися інтерфейси програмування додатків Java. Отже, щоб відбувся повний відхід від даної платформи, потрібен далеко не один рік. За цей час ви можете вивчити цю мову програмування, створити немало продуктів і заробити на своїх знаннях.

Причина #2. Java вирішує широке коло задач

Продукти Java можна зустріти буквально скрізь. Завдяки простоті та надійності Java використовують у різних сферах життя: для розробки ПЗ в державній сфері, в науці, освіті, сфері охорони здоров'я, в приватному секторі при створенні програм для трейдингу, серверні додатки для банкінгу та для багатьох інших корпоративних і ентерпрайз цілей.

Java хороша майже для усього. Навіть якщо сьогодні винайдуть достойну альтернативу, повний відхід від Java буде тривати д-у-ж-е-е-е довго років зо 20 -ть. Тому немає причин вважати, що найближчим часом ця мова програмування стане неактуальною. Переходимо до наступної причини.

Причина #3 Java продовжує активно розвиватися

Зараз актуальною є десята версія мови програмування Java. Розробка оновлень постійно триває. Актуальним залишається і старий код, тому немає потреби пристосовуватися до чогось кардинально нового.

Java 8 привнесла в світ Java концепцію функціонального програмування. Воно додало тієї самої гнучкості, за відсутність якої раніше критикували цю мову і протиставляли інші мови програмування.

Причина #4 Java мова програмування для розробки додатків під Android

Завдяки змінам у Java 5 Java 6 версіях зросла її продуктивність, і це одна з причин чому Google обрала мову програмування Java основною для розробки додатків під Android, а це однозначно підтримує Java на передовій лінії. Тому використання Java зараз важливіше, ніж коли б то не було.

Причина #5 Гарантоване працевлаштування, все тільки залежить від Вас

Стосовно працевлаштування, то дана мова програмування перевершить всі показники у порівнянні з іншими мовами. Будь який програміст або тестувальник спроможний отримати тонну вакансій і подальшу роботу, вивчаючи мову програмування Java.

Переваги мови програмування Java:

Перевага #1 Java легко вивчається новачками

Синтаксис цієї мови програмування схожий на звичайну англійську мову. В ній є мінімальна кількість складних для запам’ятовування символів. По суті, після встановлення JDK, настройки PATH і вивчення особливостей Classpath спеціаліст вже може створювати елементарні програми на Java.

Перевага #2 У Java прийняті концепції хорошого програмування

Java — об’єктно-орієнтована мова, причому це саме «об’єктне» в Java реалізоване просто-напросто на відмінно! Разом з ООП Ви вивчите

концепції наслідування, абстракції, поліморфізму і так далі. Java навчить концепціям, які можна застосовувати в більшості інших мов, наприклад, в Python

Перевага #3 Java така мова програмування рідко змінюється

Для новачків це ВЕЛИКИЙ плюс, що не потрібно відволікатися і постійно слідкувати за новими тенденціями IT. Новий функціонал Java інколи навіть занадто довго вводиться… наприклад, подібне відбувалося із замиканнями.

Перевага #4 Користувачі Java мають доступ до великої колекції бібліотек з відкритим кодом

Найкращі програмісти світу розробили шаблони, які роблять процедуру розробки більш простою. Підтримка з боку таких гігантів, як Google і Apache, дає зрозуміти, що ця мова програмування ще довго буде використовуватися в проектах. Java має гарно пророблений API, великий вибір інструментарію, велику кількість фреймворків.

Перевага #5 Java має гігантське співтовариство по всьому світу

Існує маса форумів, тематичних порталів та інших ресурсів, де обмінюються знаннями користувачі Java. Там спілкуються досвідчені програмісти, переймають досвід новачки, обговорюють новини, пов’язані з мовою програмування. Ви можете бути впевнені в тому, що ніяких проблем з пошуком інформації точно не буде.

Велика частина сучасних стартапів не використовує Java, так як на даний момент існують більш швидкі шляхи виконання того ж обсягу роботи. Проте ми навчаємо вивчати те, що приносить найбільше задоволення, так Ви станете справжнім експертом в обраній справі, зокрема кодуванні на улюбленій мові програмування.

Тому вивчайте мову Java. З нею ви зможете без проблем розробляти продукти, які будуть затребуваними в майбутньому. Java — це не важко. Спробуйте! І не припиняйте вивчати Java. Спробуйте також познайомитися із C ++ і Python — всі ці три мови схожі (об'єктно-орієнтовані імперативні). Вивчіть також JavaScript як слід (щоб побачити прототипну об'єктно-орієнтовану імперативний мову програмування) [8].

4.2. Опис процесу програмної реалізації

Тренажер запускається на виконання функцією main(String[] args).

```
public static void main(String[] args) {
    JFrame frame = new JFrame("Тренажер");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JApplet applet = new JApplet();
    frame.getContentPane().add(applet);
    applet.init();
    applet.start();
    frame.pack();
    frame.setSize(700, 730);
    frame.setResizable(false);
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
}
```

Наступним ініціалізується і виводиться аплет завдяки init(). Також підключено додаткову бібліотеку JOptionPane для відображення повідомлень.

```
import javax.swing.JOptionPane;
```

```
public void init() {
    /* Create and display the applet */
    try {
        java.awt.EventQueue.invokeAndWait(new Runnable() {
            public void run() {
                initComponents();
            }
        });
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

Створено змінну k типу int та присвоєно їй значення 1 – це крок відповідно алгоритму.

```
int k = 1;
```

Натиснувши кнопку «Довідкова інформація» виконується подія jButton2ActionPerformed, що виконує перехід до вкладки «Довідка».

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{
    if(k == 1) {
        jTabbedPane2.setSelectedIndex(0);
        jTabbedPane3.setSelectedIndex(0);
    }
    jTabbedPane1.setSelectedIndex(1);
}
```

Для кнопки «Тести» створено подію jButton3ActionPerformed. Вона відкриває вкладку «Тести». Якщо їх вже пройдено, то очищуються всі відповіді і виводиться перший крок.

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt)
{
    if(k>7) {
        k = 1;
        buttonGroup1.clearSelection();
        jComboBox1.setSelectedIndex(0);
        jComboBox2.setSelectedIndex(0);
        jComboBox3.setSelectedIndex(0);
        jComboBox4.setSelectedIndex(0);
        jComboBox5.setSelectedIndex(0);
        jComboBox6.setSelectedIndex(0);
        jComboBox7.setSelectedIndex(0);
        jComboBox8.setSelectedIndex(0);
        jComboBox9.setSelectedIndex(0);
        jComboBox10.setSelectedIndex(0);
        jComboBox11.setSelectedIndex(0);
        jComboBox12.setSelectedIndex(0);
        jComboBox13.setSelectedIndex(0);
        jComboBox14.setSelectedIndex(0);
        jComboBox16.setSelectedIndex(0);
        jComboBox17.setSelectedIndex(0);
        jComboBox18.setSelectedIndex(0);
        jComboBox19.setSelectedIndex(0);
        jComboBox20.setSelectedIndex(0);
        jComboBox21.setSelectedIndex(0);
        jComboBox22.setSelectedIndex(0);
        jComboBox23.setSelectedIndex(0);
```

```
            jComboBox24.setSelectedIndex(0);
            jComboBox25.setSelectedIndex(0);
        }
        jTabbedPane1.setSelectedIndex(2);
        jTabbedPane3.setSelectedIndex(k-1);
    }
```

Основні дії виконуються подією jButton1ActionPerformed для кнопки «Далі». Тут відбувається перевірка вибраної відповіді відповідно поточного кроку (див. Додаток А).

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
        switch(k) {
            case 1:
                if(jRadioButton1.isSelected()) {
                    jTabbedPane3.setSelectedIndex(k);
                    k++;
                } else {
                    jTabbedPane1.setSelectedIndex(1);
                    jTabbedPane2.setSelectedIndex(0);
                }
                break;
            case 7:
                if(jComboBox22.getSelectedIndex()==1 &&
jComboBox23.getSelectedIndex()==2
                        && jComboBox24.getSelectedIndex()==0 &&
jComboBox25.getSelectedIndex()==3) {
                    JOptionPane.showMessageDialog(jTabbedPane1,
"<html>Всі тести тренажеру пройдено", "Вітаємо!",
JOptionPane.INFORMATION_MESSAGE);
```

```
            jTabbedPane1.setSelectedIndex(0);
            k++;
        } else {
            jTabbedPane1.setSelectedIndex(1);
            jTabbedPane2.setSelectedIndex(4);
        }
        break;
    }
```

4.3. Необхідна користувачу програми інструкція

Тренажер поділений на вкладки (рис. 4.1):

- Головна;
- Довідка;
- Тести.

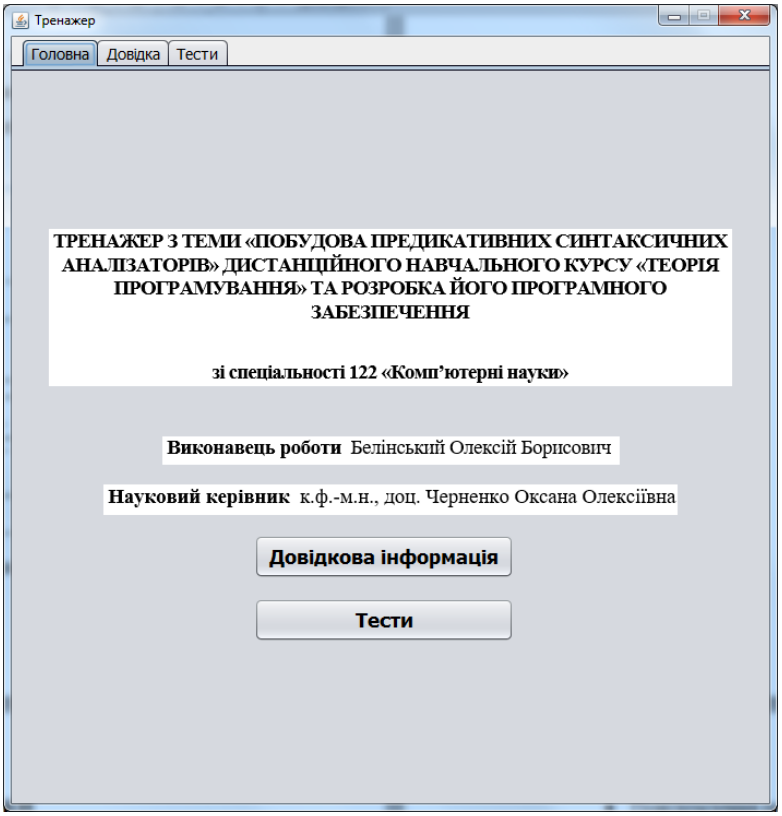

Рисунок 4.1 – Вкладка «Головна»

Щоб переглянути довідкову інформацію потрібно натиснути кнопку «Довідкова інформація» або просто вибрати вкладку «Довідка». В ній пропонується наступний матеріал:

- робота програми аналізатора;
- алгоритм 1. Нерекурсивний предиктивний аналіз;
- алгоритм 2. Побудова множини FIRST для символів граматики;
- алгоритм 3. Побудова FOLLOW(*X*) для всіх *X*-нетерміналів граматики;
- алгоритм 4. Побудова таблиць предикативного аналізу.

Якщо матеріалу багато, то повністю його можна переглянути за допомогою полоси прокрутки (рис. 4.2). В іншому разі її не буде (рис. 4.3).

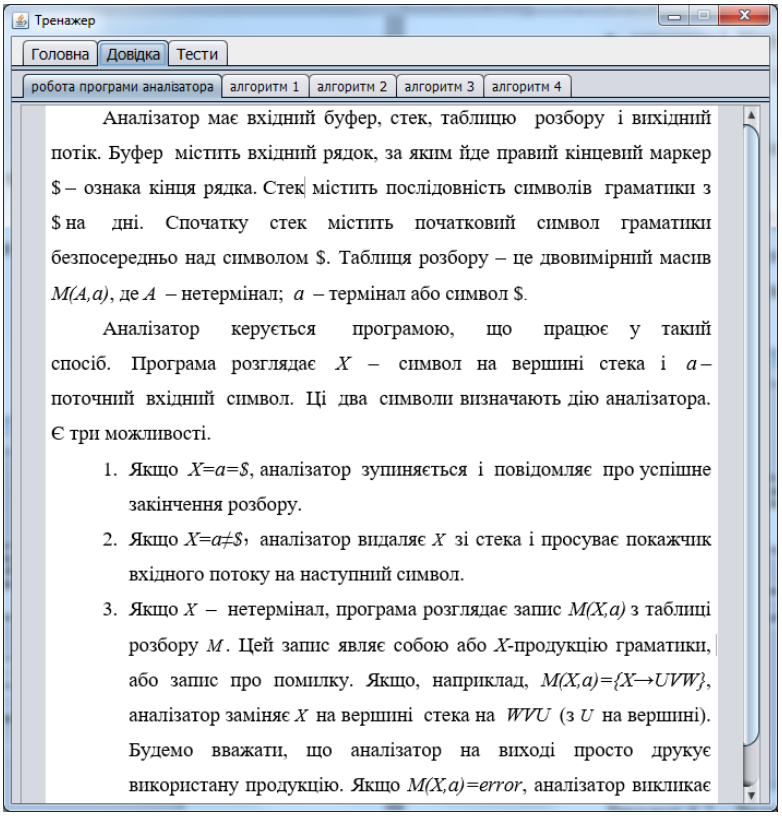

Рисунок 4.2 – Вкладка «робота програми аналізатора»

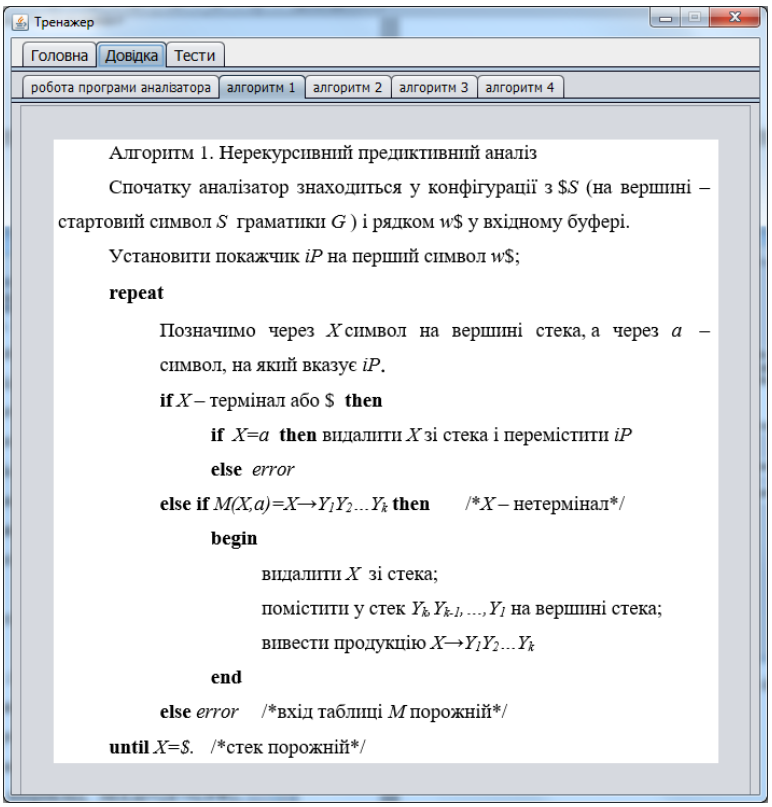

Рисунок 4.3 – Вкладка «алгоритм 1»

Для переходу до тестів слід натиснути кнопку «Тести» або вибрати вкладку «Тести». При цьому відображається завдання першого кроку і варіанти відповіді (рис. 4.4).

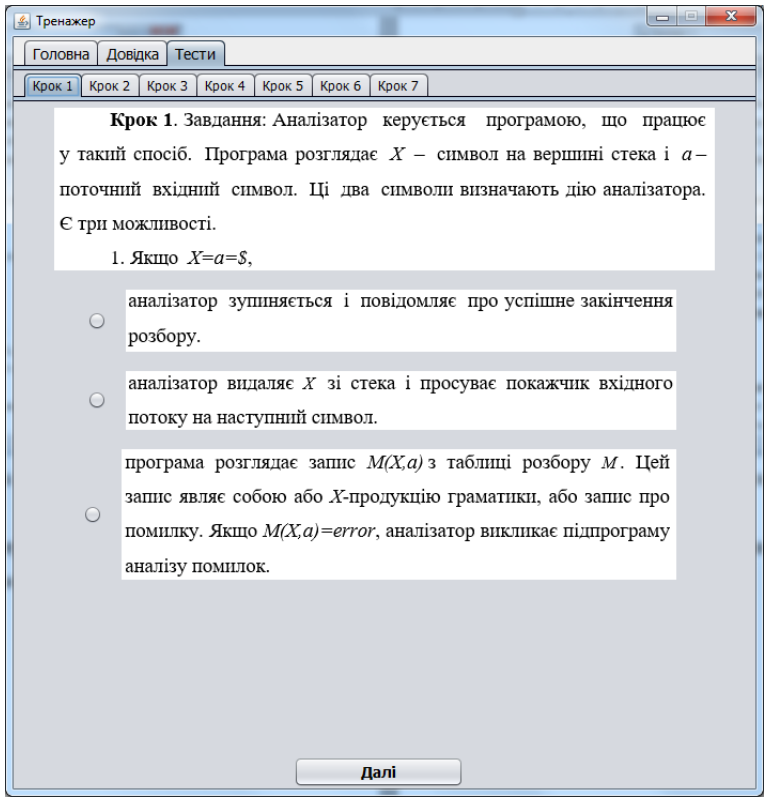

Рисунок 4.4 – Вкладка «Крок 1»

Потрібно вибрати один із варіантів і натиснути «Далі». Якщо відповідь правильна, то виведеться наступний крок (рис. 4.5). Якщо ні – довідка до цього завдання (рис. 4.2).

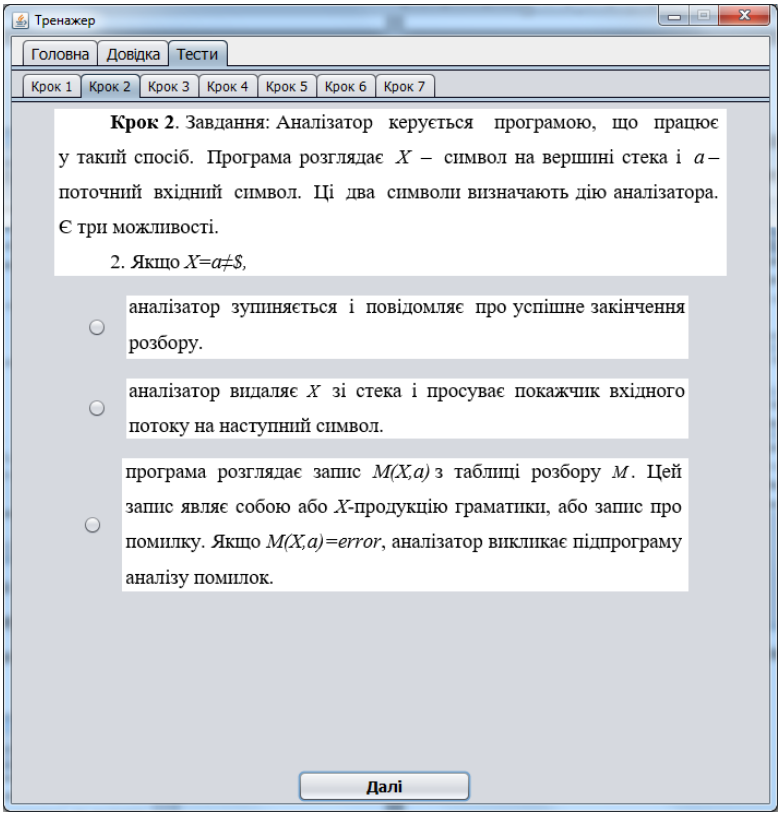

Рисунок 4.5 – Вкладка «Крок 2»

На четвертому кроці з’явиться завдання, де необхідно встановити послідовність (рис. 4.6). Для кожного рядка потрібно вибрати номер зі списку, що відкривається (рис. 4.7).

В даному випадку лише якщо все встановлено вірно, то відбудеться перехід до наступного кроку.

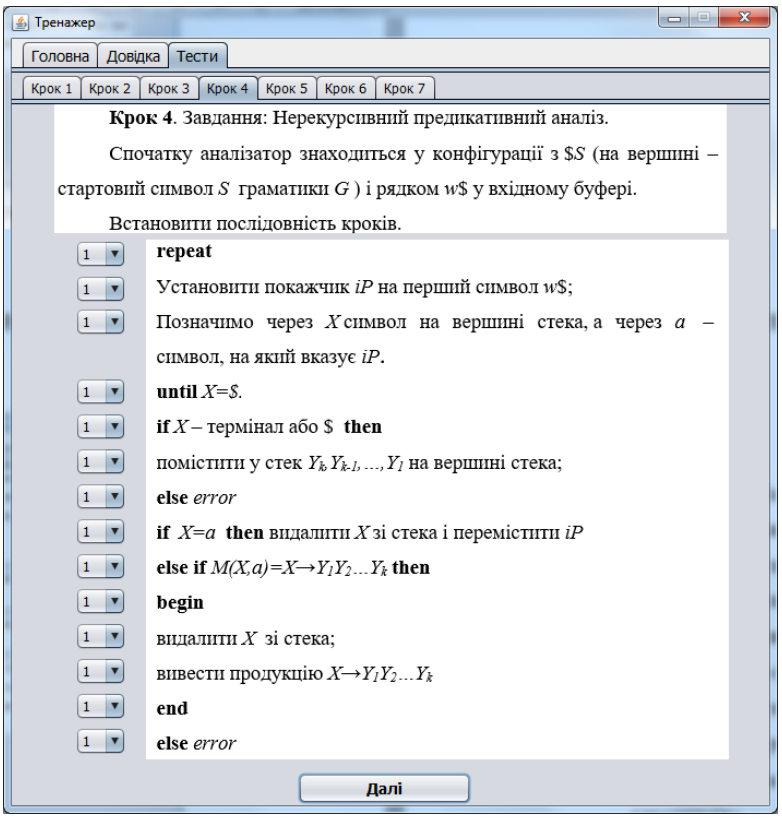

Рисунок 4.6 – Вкладка «Крок 4»

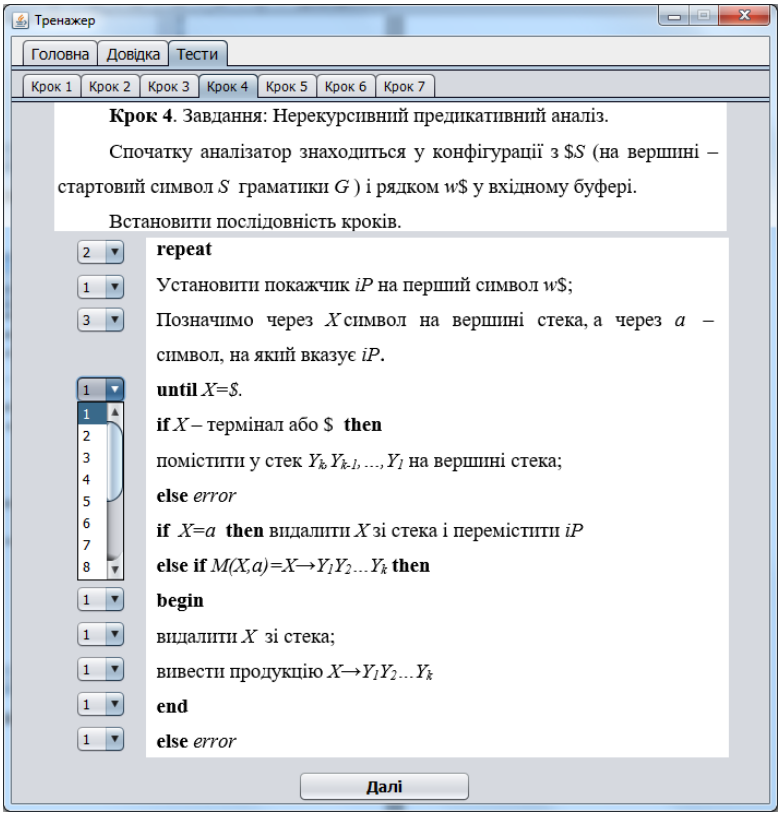

Рисунок 4.7 – Вкладка «Крок 4», встановлення відповідності

Після відповіді на останній крок виведеться повідомлення про завершення (рис. 4.8). Потім відбудеться перехід до вкладки «Головна», де можна знову переглянути довідку або перейти до тестів і почати їх проходження заново.

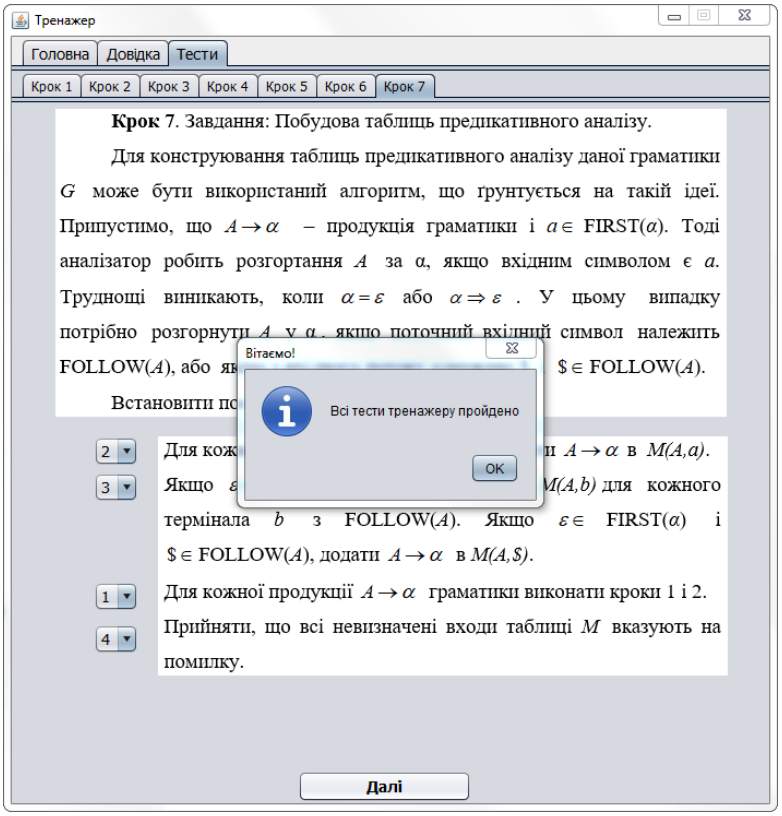

Рисунок 4.8 – Вкладка «Крок 7», завершення проходження

ВИСНОВКИ

Метою виконаної роботи є алгоритм роботи тренажеру та тренажер з теми «Побудова предикативних синтаксичних аналізаторів» дистанційного навчального курсу «Теорія програмування».

Було розглянуто декілька тренажерів:

- тренажер «Синтаксичний аналіз» (дисципліна «Теорія програмування»);
- тренажер з теми «Рекурсивні функції» дистанційного навчального курсу «Теорія алгоритмів»;
- тренажер з теми «Нормальні алгоритми» з дисципліни «Теорія алгоритмів»

По кожному з них описано їх принцип роботи.

В алгоритмі виведено основні кроки побудови предикативних синтаксичних аналізаторів для подальшої програмної реалізації.

Наведено перелік основних завдань і довідкової інформації по ним:

- робота програми аналізатора;
- алгоритм 1. Нерекурсивний предиктивний аналіз;
- алгоритм 2. Побудова множини FIRST для символів граматики;
- алгоритм 3. Побудова FOLLOW(X) для всіх X-нетерміналів граматики;
- алгоритм 4. Побудова таблиць предиктивного аналізу.

Після проходження виводиться повідомлення про завершення, відкривається стартова сторінка.

Тренажер містить такі елементи:

- Головне вікно;
- Ознайомчий матеріал за темою;
- Тести;
- Довідкову інформацію;

- Повідомлення про завершення.

При неправильній відповіді реалізовано виведення довідкової інформації.

Результати роботи впроваджені в дистанційному навчальному курсі «Теорія програмування».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ємець О. О. Методичні рекомендації до виконання бакалаврської роботи для студентів за освітньою програмою «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки та інформаційні технології» галузь знань - 12 «Інформаційні технології» / О.О.(Олег) Ємець. – Полтава : РВВ ПУЕТ, 2017. – 71 с.
2. Морзе Н.В. Інформаційні технології в навчанні : навч. посіб. / за ред. Н.В. Морзе. – К. : Видавнича група BHV, 2004. – 240 с.
3. Воронкін О.С. Організація дистанційних технологій навчання на основікомп'ютерних інформаційних систем вищих навчальних закладів України [Електронний ресурс]. – Режим доступу: http://www.nbuv.gov.ua/ejournals/vsunud/2009-6E/09vosnzu.htm.
4. Черненко О.О. Електронний навчально-методичний посібник для самостійного вивчення навчальної дисципліни «Теорія програмування» для студентів напряму 6.040302 «Інформатика» – Режим доступу: http://tprogr.ho.ua.
5. Бабій М.С. Теорія програмування: Навчальний посібник [Електронний ресурс] / М.С. Бабій, О.П. Чекалов.– Суми: Вид-во СумДУ, 2009. – 181 с.
6. Нікітченко М.С. Теоретичні основи програмування: Навчальний посібник [Електронний ресурс] / М.С. Нікітченко. – Київ: КНУ ім. Т.Г. Шевченка, 2009. – 200 с. – Режим доступу: http://ttp.unicyb.kiev.ua/doc/TOP.pdf.
7. Кейденхед Роджерс Java за 24 часа / 8-е издание / Кейденхед Роджерс. – Диалектика, 2019. – 480 с.
8. Блох Дж. Java: эффективное программирование / Блох Дж. – Диалектика, 2019. – 464 с.
9. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання: ДСТУ 7.1-2006. – [Чинний від 2007-07-01]. – К. : Держспоживстандарт України, 2007. – 47 с.

ДОДАТОК А. КОД ПРОГРАМИ