ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ БІЗНЕСУ ТА СУЧАСНИХ ТЕХНОЛОГІЙ

ФОРМА НАВЧАННЯ ДЕННА
КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА СОЦІАЛЬНОЇ ІНФОРМАТИКИ

Допускається до захисту

Завідувач кафедри _____О.О. Ємець
(підпис)

«_____»_____2021 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО БАКАЛАВРСЬКОЇ РОБОТИ

на тему
Розробка програмного забезпечення тренажеру з теми «Алгоритм видалення лівої рекурсії з граматики» дистанційного навчального курсу «Теорія програмування»

зі спеціальності 122 «Комп'ютерні науки»

Виконавець роботи Пластуняк Руслан Богданович
_____ «___»_____2021р.
(підпис)
Науковий керівник к.ф.-м.н., доц. Черненко Оксана Олексіївна
_____ «___»_____2021р.
(підпис)

ПОЛТАВА 2021 р.

ЗМІСТ

ВСТУП

Сучасний рівень розвитку комп’ютерної техніки та різноманітного програмного забезпечення надає широкі можливості для підвищення ефективності навчання. Використання комп’ютерних технологій сприяє удосконаленню системи освіти та забезпеченню якісно нового її рівня.

Поняття “дистанційне навчання”, “дистанційна освіта” трактуються по- різному.

Під “дистанційним навчанням” розуміють індивідуальний процес набуття знань, умінь, навичок і способів пізнавальної діяльності людини, який відбувається в основному за опосередкованої взаємодії віддалених один від одного учасників навчального процесу у спеціалізованому середовищі, яке функціонує на базі сучасних психолого-педагогічних та інформаційно- комунікаційних технологій.

Метою дистанційного навчання є надання освітніх послуг шляхом застосування у навчанні сучасних інформаційно-комунікаційних технологій за певними освітніми або освітньо-кваліфікаційними рівнями відповідно до державних стандартів освіти; за програмами підготовки громадян до вступу у навчальні заклади, підготовки іноземців та підвищення кваліфікації працівників.

Метою роботи є розробка тренажеру з теми «Алгоритм видалення лівої рекурсії з граматики» дистанційного навчального курсу «Теорія програмування».

Об’єктом розробки є елементи тренажеру з дисципліни.

Предметом розробки є тренажер з теми «Алгоритм видалення лівої рекурсії з граматики».

Перелік використаних методів: мова програмування Java, середовище програмування NetBeans.

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

Робота складається з чотирьох розділів.

Обсяг пояснювальної записки: 36 стор., в т.ч. основна частина - 30 стор., джерела - 6 назв.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

Рекурсія є одним з фундаментальних методів програмування. При розробці програми часто необхідно звести початкові завдання до простіших.

Спосіб зведення завдання до нього ж самого, але із зміненими початковими даними, називається **рекурсією.**

Обчислення факторіалу $n!=1\cdot 2\cdot 3\ldots n$ зводиться до обчислення добутку цілих чисел від 1 до n : $0!=1,\ 1!=1,\ \ 4!=1\cdot 2\cdot 3\cdot 4=24$. Такий добуток можна легко обчислити за допомогою циклічних конструкцій, наприклад, оператора циклу for. Проте, $n!$ рівне n , помноженому на добуток від 1 до $(n-1)$, а саме, $n!=(n-1)!\cdot n,\ \ 0!=1$ за означенням $4!=3!\cdot 4,\ 6!=5!\cdot 6$.

В цьому випадку означуване поняття $n!$ описується через саме себе (n) і через простіше поняття $(n-1)!$. Щоб знайти $n!$, треба застосувати функцію факторіала до числа $(n-1)$, потім до $(n-2)$ і т.д. аж до 1. Це і означає, що таке означення факторіалу рекурсивне.

Зміст і потужність рекурсивного визначення, а також його головне призначення полягає в тому, що воно дозволяє за допомогою кінцевого виразу визначити нескінчену множину об’єктів.

Програми, в яких використовують рекурсивні процедури відрізняються простотою, наочністю і комплектністю тексту, оскільки рекурсивна процедура указує, що потрібно робити, а нерекурсивна більше акцентує увагу на тому, як потрібно робити. Проте за цю простоту доводиться розплачуватися неекономним використанням оперативної пам’яті, оскільки при кожному рекурсивному виклику для локальних змінних, а також для параметрів процедури, які передаються за значенням, виділяються нові комірки.

Рекурсивний спуск — алгоритм синтаксичного аналізу, будується на основі взаємно рекурсивних процедур (або не рекурсивних еквівалентів), кожна із яких реалізує одну із продукцій граматики.

Розбір рекурсивним спуском приховує небезпеку зациклювання. Це можливо, якщо продукція має вигляд <вир> -> <вир> <терм>, через те, що нетермінал <вир> ліворекурсивний.

Ліворекурсивним назвемо нетермінал А, якщо в граматиці є продукція виду $A \to A\alpha$.

Якщо алгоритм обере цю продукцію, то, оскільки права частина починається з <вир>, буде знову викликана процедура для нетермінала <вир>. Поточний символ, що керує процесом розбору, може змінитися тюльки, якщо буде виявлений його збіг з початковим терміналом у правій частині продукції. У нашому випадку продукція починається з нетермінала і поточний символ залишається незмінним, що викликає зациклювання.

Ліву рекурсію можна усунути переписуванням небезпечної продукції.

Ліворекурсивна пара продукцій $A \to A\alpha \mid \beta$ може бути замінена неліворекурсивними продукціями:

$$A \to \beta A',\ A' \to \alpha A' \mid \varepsilon,$$

(1.1)

де α та β - ланцюжки терміналів і нетерміналів, що не починаються з *A*, *A'* - новий нетермінал. Продукції виду (1.1) називаються **праворекурсивними**.

Приклад 1.

Розглянемо граматику для арифметичних виразів з продукціями:

$E \to E + T \mid T$,

$T \to T * F \mid F$,

$F \to (E) \mid id$.

Після видалення рекурсії з продукцій для *E* і *T* одержимо:

$E \to TE',\quad E' \to +TE' \mid \varepsilon$,

$T \to FT',\qquad T' \to *FT' \mid \varepsilon$,

$F \to (E) | id$.

Неважливо, яка загальна кількість *A*-продукцій. Ми можемо видалити безпосередню ліву рекурсію з них за допомогою такої технології. Спочатку групуємо *A*-продукції:

$A \to A\alpha_1 | A\alpha_2 | \ldots | A\alpha_m | \beta_1 | \ldots | \beta_n$,

де β_i не починається з *A*. Потім заміняємо ці *A*-продукції на

$A \to \beta_1 A' | \beta_2 A' | \cdots | \beta_n A'$, $A' \to \alpha_1 A' | \alpha_2 A' | \cdots | \alpha_m A' | \varepsilon$. (1.2)

Нетермінал *A* породжує ті самі рядки, що і раніше, але тепер немає лівої рекурсії. За допомогою цієї процедури видаляються всі безпосередні ліві рекурсії, але не видаляється ліва рекурсія, що включає два або більше кроків породження.

Приклад 2.

Розглянемо граматику з продукціями:

$S \to A\alpha | b$;

$A \to Ac | Sd | \varepsilon$.

Нетермінал *S* ліворекурсивний, оскільки $S \Rightarrow Aa \Rightarrow Sda$, але ця рекурсія не є безпосередньою.

Наведений нижче алгоритм дозволяє видалити всі ліві рекурсії з граматики.

Алгоритм видалення лівої рекурсії

Крок 1 Упорядковуємо нетермінали в довільному порядку $A_1, A_2, \ldots, A_n$.

Крок 2

for $i := 1$ *to* n *do begin*

for $j := 1$ *to* $i-1$ *do begin*

Замінити всі продукції вигляду $A_i \to A_j\gamma$

продукціями $A_i \to \delta_j\gamma | \delta_2\gamma | \cdots | \delta_k\gamma$,

де $A_j \to | \delta_2 | \cdots | \delta_k$, – всі поточні A_j-продукції

end

Видалити безпосередню ліву рекурсію серед A_i-продукцій за формулами (1.2)

end

Обґрунтування працездатності алгоритму базується на тому, що після $(i-1)$-ої ітерації зовнішнього циклу **for** кроку 2 для будь-якої продукції вигляду $A_k \to A_l a,\ k < i$ повинно бути $l > k$. У результаті на наступній ітерації внутрішнього циклу (за j) послідовно збільшується нижня границя m всіх продукцій $A_i \to A_m a$, поки не буде досягнуто $m \geq i$. Потім, видаляючи безпосередню ліву рекурсію для A_i-продукцій, одержуємо $m > i$.

Розглянутий алгоритм гарантовано працює з граматиками, які не мають циклів (породжень типу $A \Rightarrow A$) і ε-продукцій (продукцій типу $A \to \varepsilon$). Як цикли, так і ε-продукції, можуть бути вилучені попередньо за формулами [2, 3].

Приклад 3.

Дана граматика:

$A \to S\alpha$

$S \to S\beta \mid A\gamma \mid \beta$

Серед правил A безпосередній рекурсії немає, тому під час першої ітерації зовнішнього циклу нічого не відбувається. Під час другої ітерації зовнішнього циклу правило $S \to A\gamma$ переходить в $S \to S\alpha\gamma$.

Граматика має вигляд

$A \to S\alpha$

$S \to S\beta \mid S\alpha\gamma \mid \beta$

Усуваємо ліву рекурсію для S

$S \to \beta S_1$

$S_1 \to \beta S_1 \mid \alpha\gamma S_1 \mid \beta \mid \alpha\gamma$ [4].

2. ТЕОРЕТИЧНА ЧАСТИНА

2.1. Алгоритмізація задачі за темою роботи

Користувачу поетапно виводиться спочатку матеріал, а потім розв'язок прикладу на основі розглянутої інформації.

Крок 1. Матеріал для ознайомлення: «**Рекурсивний спуск** — алгоритм синтаксичного аналізу, будується на основі взаємно рекурсивних процедур (або не рекурсивних еквівалентів), кожна із яких реалізує одну із продукцій граматики.

Ліворекурсивним назвемо нетермінал А, якщо в граматиці є продукція виду $A \to A\alpha$.

Ліву рекурсію можна усунути переписуванням небезпечної продукції.

Ліворекурсивна пара продукцій $A \to A\alpha \mid \beta$ може бути замінена неліворекурсивними продукціями:

$$A \to \beta A',\ A' \to \alpha A' \mid \varepsilon, \qquad (1)$$

де α та β - ланцюжки терміналів і нетерміналів, що не починаються з A, A' - новий нетермінал. Продукції виду (1) називаються **праворекурсивними**.».

Перехід на наступний крок.

Крок 2. Умова прикладу: «Розглянемо граматику для арифметичних виразів з продукціями:

$E \to E + T \mid T$,

$T \to T * F \mid F$,

$F \to (E) \mid id$.

Після видалення рекурсії з продукцій для E і T одержимо: ».

Розв'язання:

$E \to$ [ ]

Якщо відповідь – $E \to TE'$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 3. Умова прикладу залишається.

Розв'язання:

$E \to TE',$

$E' \to \boxed{} | \varepsilon$

Якщо відповідь – $E' \to +TE' | \varepsilon$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 4. Умова прикладу залишається.

Розв'язання:

$E \to TE', \quad E' \to +TE' | \varepsilon,$

$T \to \boxed{}$

Якщо відповідь – $T \to FT'$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 5. Умова прикладу залишається.

Розв'язання:

$E \to TE', \quad E' \to +TE' | \varepsilon,$

$T \to FT'$

$T' \to \boxed{} | \varepsilon$

Якщо відповідь – $T' \to *FT' | \varepsilon$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 6. Умова прикладу залишається.

Розв'язання:

$E \to TE', \quad E' \to +TE' | \varepsilon,$

$T \to FT', \qquad T' \to *FT' | \varepsilon,$

$F \to \boxed{} | id$

Якщо відповідь – $F \to (E)|id$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 7. Матеріал для ознайомлення: «Неважливо, яка загальна кількість *A*-продукцій. Ми можемо видалити безпосередню ліву рекурсію з них за допомогою такої технології. Спочатку групуємо *A*-продукції:

$$A \to A\alpha_1 | A\alpha_2 | \ldots | A\alpha_m | \beta_1 | \ldots | \beta_n,$$

де β_i не починається з *A*. Потім заміняємо ці *A*-продукції на

$$A \to \beta_1 A' | \beta_2 A' | \cdots | \beta_n A' | \ , \quad A' \to \alpha_1 A' | \alpha_2 A' | \cdots | \alpha_m A' | \varepsilon \ . \qquad (2)$$

Нетермінал *A* породжує ті самі рядки, що і раніше, але тепер немає лівої рекурсії. За допомогою цієї процедури видаляються всі безпосередні ліві рекурсії, але не видаляється ліва рекурсія, що включає два або більше кроків породження.».

Перехід на наступний крок.

Крок 8. Умова прикладу: «Розглянемо граматику з продукціями:

$S \to A\alpha | b;$

$A \to Ac | Sd | \varepsilon$.».

Розв'язання:

Нетермінал *S*

- ліворекурсивний.
- праворекурсивний.

Якщо відповідь – ліворекурсивний, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 9. Умова прикладу залишається.

Розв'язання:

Нетермінал *S* ліворекурсивний, оскільки

- $S \Rightarrow Aa \Rightarrow Sda$.
- $S \Rightarrow Aa \Rightarrow Aca$.

Якщо відповідь – $S \Rightarrow Aa \Rightarrow Sda$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 10. Умова прикладу залишається.

Розв'язання:

Нетермінал S ліворекурсивний, оскільки $S \Rightarrow Aa \Rightarrow Sda$, але ця рекурсія

- є безпосередньою.
- не є безпосередньою.

Якщо відповідь – не є безпосередньою, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 11. Матеріал для ознайомлення: «**Алгоритм видалення лівої рекурсії**

Крок 1 Упорядковуємо нетермінали в довільному порядку $A_1, A_2, \ldots, A_n$.

Крок 2

for $i := 1$ *to* n *do begin*

for $j := 1$ *to* $i-1$ *do begin*

Замінити всі продукції вигляду $A_i \to A_j\gamma$

продукціями $A_i \to \delta_j\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_к\gamma,$

де $A_j \to \mid \delta_2 \mid \cdots \mid \delta_к,$ – всі поточні A_j-продукції

end

Видалити безпосередню ліву рекурсію серед A_i-продукцій за формулами $A \to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \mid$, $A' \to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \varepsilon$

end».

Перехід на наступний крок.

Крок 12. Обґрунтування працездатності алгоритму:

Базується на тому, що після $(i-1)$-ої ітерації зовнішнього циклу **for** кроку 2 для будь-якої продукції вигляду $A_k \to A_l a,\ k < i$ повинно бути

- $l < k$.
- $l > k$.

Якщо відповідь – $l > k$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 13. Обґрунтування працездатності алгоритму:

У результаті на наступній ітерації внутрішнього циклу (за j) послідовно збільшується нижня границя m всіх продукцій $A_i \to A_m a$, поки не буде досягнуто

- $m \geq i$.
- $m > i$.

Якщо відповідь – $m \geq i$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 14. Обґрунтування працездатності алгоритму:

Потім, видаляючи безпосередню ліву рекурсію для A_i-продукцій, одержуємо

- $m \geq i$.
- $m > i$.

Якщо відповідь – $m > i$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 15. Обґрунтування працездатності алгоритму:

Розглянутий алгоритм гарантовано працює з граматиками, які не мають

- циклів (породжень типу $A \Rightarrow A$).
- ε-продукцій (продукцій типу $A \to \varepsilon$).
- циклів (породжень типу $A \Rightarrow A$) і ε-продукцій (продукцій типу $A \to \varepsilon$)

Якщо відповідь – циклів (породжень типу $A \Rightarrow A$) і ε-продукцій (продукцій типу $A \to \varepsilon$), то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 16. Умова прикладу: «Дана граматика:

$A \to S\alpha$

$S \to S\beta \mid A\gamma \mid \beta$».

Розв’язання:

Під час першої ітерації зовнішнього циклу

- нічого не відбувається.
- правило $S \to A\gamma$ переходить в $S \to S\alpha\gamma$.

Якщо відповідь – нічого не відбувається, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 16. Умова прикладу залишається.

Розв’язання:

Під час другої ітерації зовнішнього циклу

- нічого не відбувається.
- правило $S \to A\gamma$ переходить в $S \to S\alpha\gamma$.

Якщо відповідь – правило $S \to A\gamma$ переходить в $S \to S\alpha\gamma$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 17. Умова прикладу залишається.

Розв’язання:

Граматика має вигляд

$A \to$ ☐

Якщо відповідь – $A \to S\alpha$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 18. Умова прикладу залишається.

Розв’язання:

Граматика має вигляд

$A \to S\alpha$

$S \to$ ☐

Якщо відповідь – $S \to S\beta \mid S\alpha\gamma \mid \beta$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 19. Умова прикладу залишається.

Розв’язання:

Усуваємо ліву рекурсію для S

$S \rightarrow$ ☐

Якщо відповідь – $S \rightarrow \beta S_1$, то перехід на наступний крок. Якщо ні, то виводиться повідомлення про помилку.

Крок 20. Умова прикладу залишається.

Розв’язання:

Усуваємо ліву рекурсію для S

$S \rightarrow \beta S_1$

$S_1 \rightarrow$ ☐

Якщо відповідь – $S_1 \rightarrow \beta S_1 \mid \alpha\gamma S_1 \mid \beta \mid \alpha\gamma$, то кінець. Якщо ні, то виводиться повідомлення про помилку.

2.2. Розробка блок-схеми, яка підлягає програмуванню

На рисунку 2.1 зображено блок-схему алгоритму роботи тренажеру.

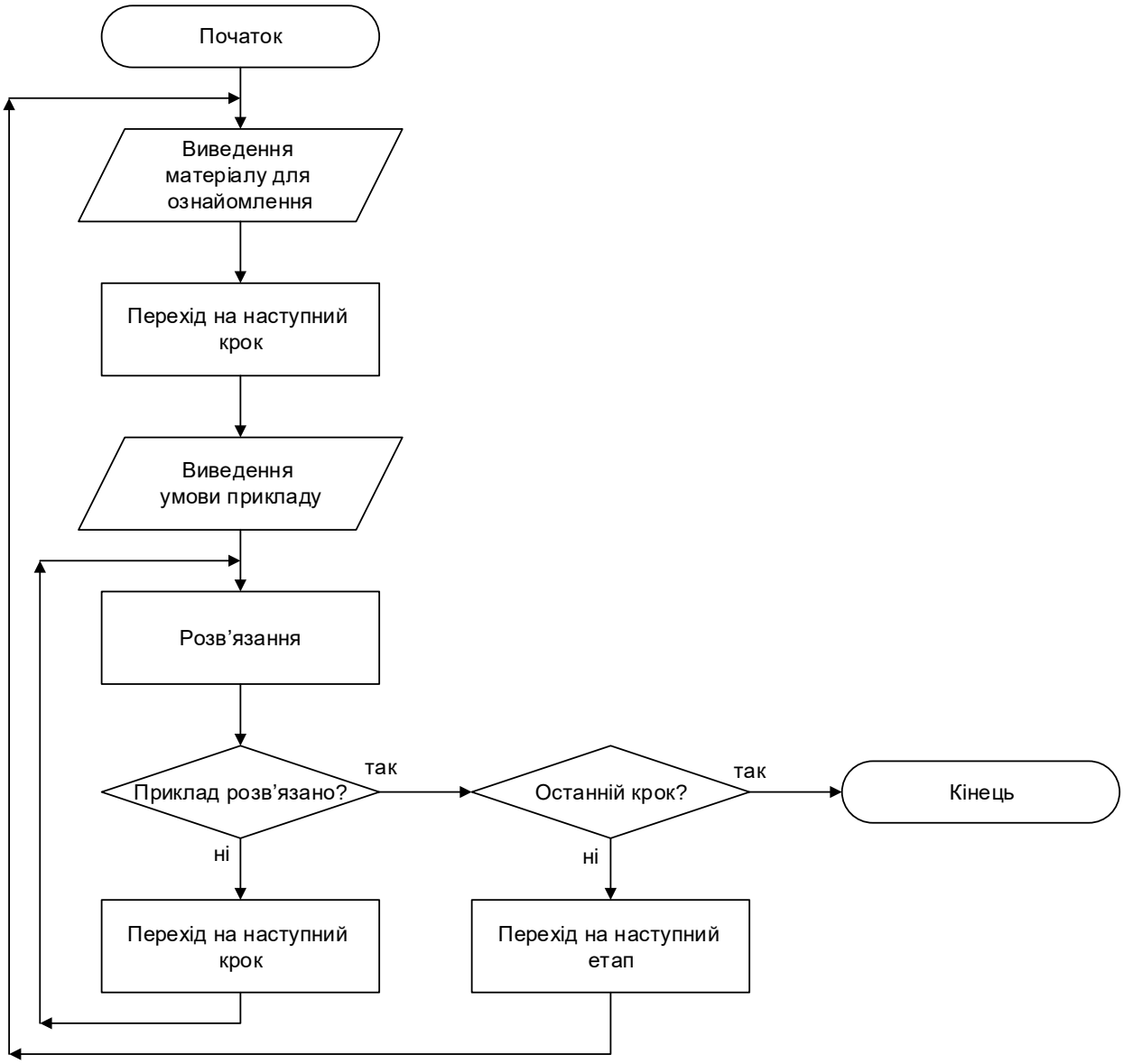

Рисунок 2.1 – Блок-схема алгоритму роботи тренажеру

На рисунку 2.2 зображено блок-схему перевірки відповіді при розв’язанні прикладів.

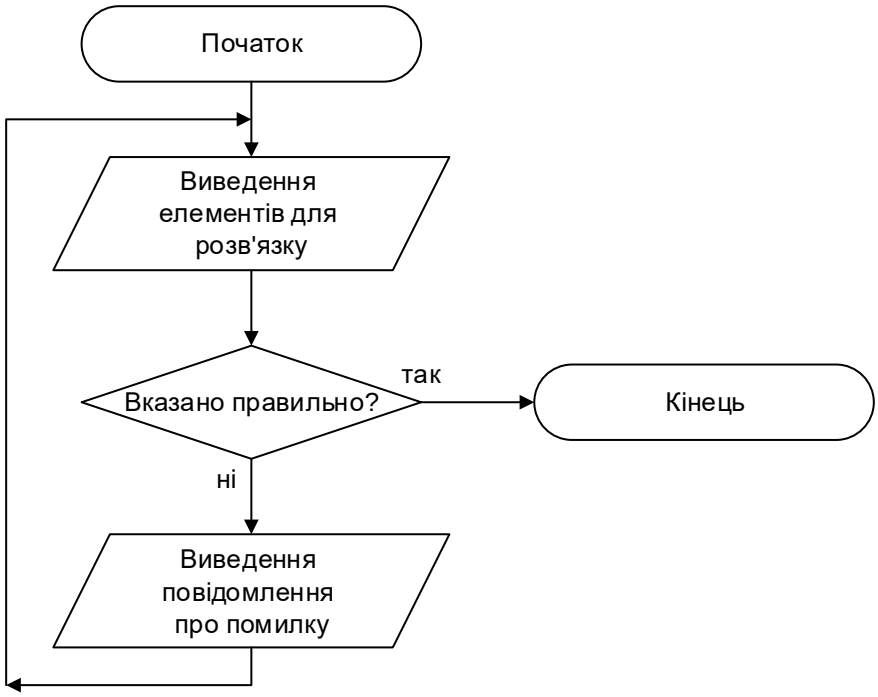

Рисунок 2.2 – Блок-схема перевірки відповіді при розв’язанні прикладів

2.3. Опис мови програмування

Java працює на 3 мільярдах пристроїв по всьому світу – це ПК, мобільна електроніка, пристрої що носяться, бортові системи, мережеве обладнання, промислове програмоване обладнання.

На основі Java технологій (точніше, в основному на основі J2EE-технологій) реалізовані такі проекти світового рівня, як Amazon, eBay, Twitter, Yandex, LinkedIn, Yahoo!, OpenOffice, RuneScape. Продукти компанії Google також багато в чому спираються на технології Java.

Java підтримує всі новітні технології (в тому числі web- та android-розробки) і містить засоби, що значно спрощують і прискорюють процес розробки. Крім того, мова активно розвивається, а проблеми продуктивності зведені до мінімуму.

Відомо, що програма, написана на будь-якій мові високого рівня, не може бути відразу ж виконана. Її спочатку треба компілювати, тобто перевести в послідовність машинних команд процесора – об'єктний модуль. Але і він, як правило, не може бути відразу ж виконаний: об'єктний модуль

треба ще скомпонувати з бібліотеками використаних в модулі функцій і розв'язати перехресні посилання між секціями об'єктного модуля, отримавши в результаті завантажувальний модуль – повністю готову до виконання програму (рис. 1.1).

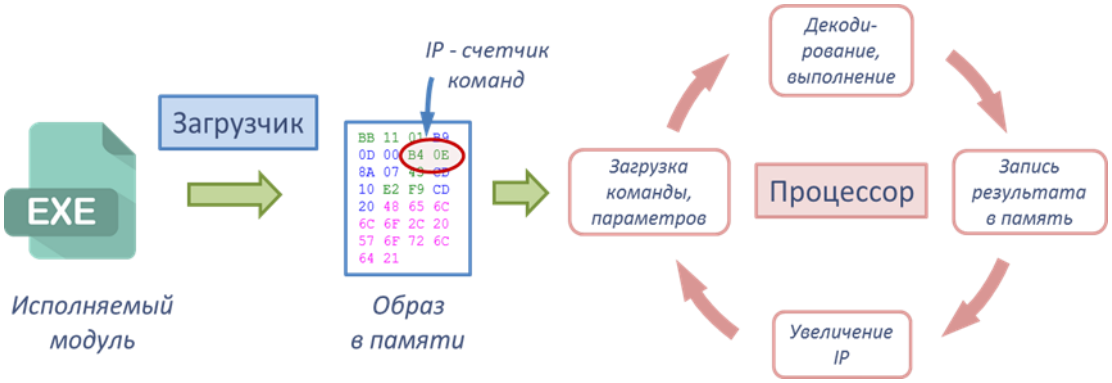

Рисунок 1.1 – Процес виконання звичайної програми

Java-програма не може уникнути цих процедур, але тут проявляється головна особливість технології Java – програма компілюється не в команди якогось конкретного процесора, а в команди так званої віртуальної машини Java (JVM, Java Virtual Machine). Віртуальна машина Java – це сукупність команд разом з системою їх виконання. Віртуальна машина Java повністю стекова, використовує плоску адресацію пам'яті і невелику кількість регістрів. Тому команди JVM короткі, більшість з них має довжину 1 байт, а середня довжина команди складає 1,8 байти: ось чому команди JVM називають байт-кодами (bytecodes).

Повний опис команд і всієї архітектури JVM міститься в специфікації віртуальної машини Java (VMS, Virtual Machine Specification). Ця специфікація гарантує однакове виконання java-коду на будь-якій апаратній платформі. Так реалізується принцип Java "Write once, run anywhere"– "Написано один раз, виконується де завгодно".

Таким чином, програми на Java транслюються в байт-код, що виконується віртуальною машиною Java (JVM) – програмою-інтерпретатором, яка обробляє команди байт-коду і перетворює їх в інструкції конкретного процесора або виклики функцій конкретної

операційної системи (рис. 1.2). Такі віртуальні Java-машини розроблені для всіх популярних операційних систем, а також для багатьох маловідомих.

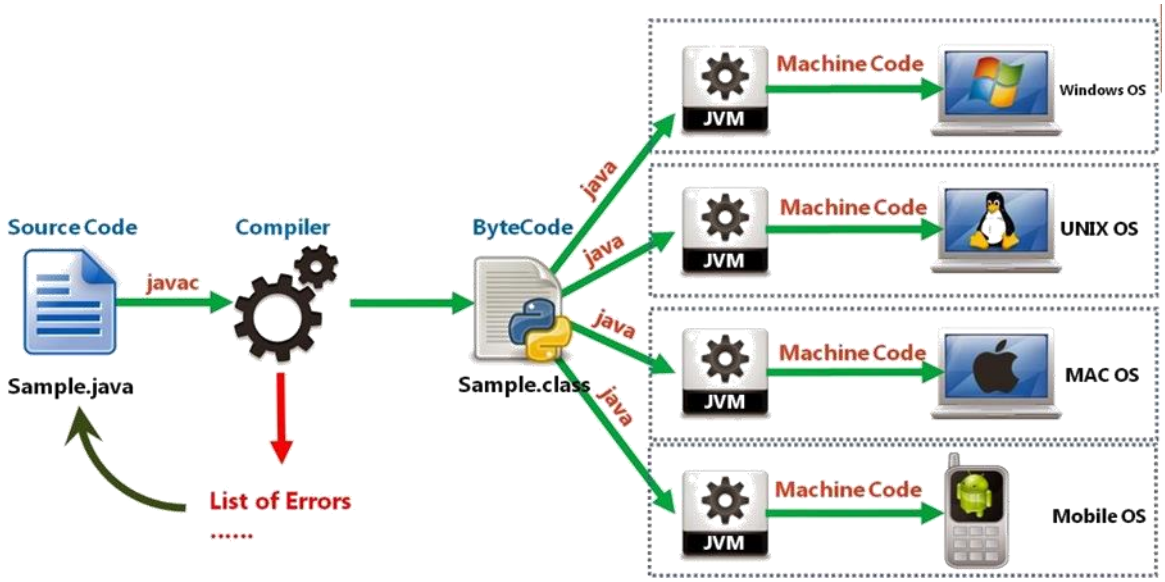

Рисунок 1.2 – Виконання Java-програми

Мова Java володіє всіма перевагами сучасної мови програмування.

Ідеї, закладені в концепцію віртуальної машини Java, надихнули безліч ентузіастів на розширення переліку мов, що виконуються на віртуальній машині. Ці ідеї знайшли також вираз в специфікації загальномовної інфраструктури CLI (Common Language Infrastructure), закладеної в основу платформи .NET компанією Microsoft [5].

3. ПРАКТИЧНА ЧАСТИНА

3.1. Опис процесу програмної реалізації

Після створення проекту потрібно на робоче вікно додати панель (рис. 2.3).

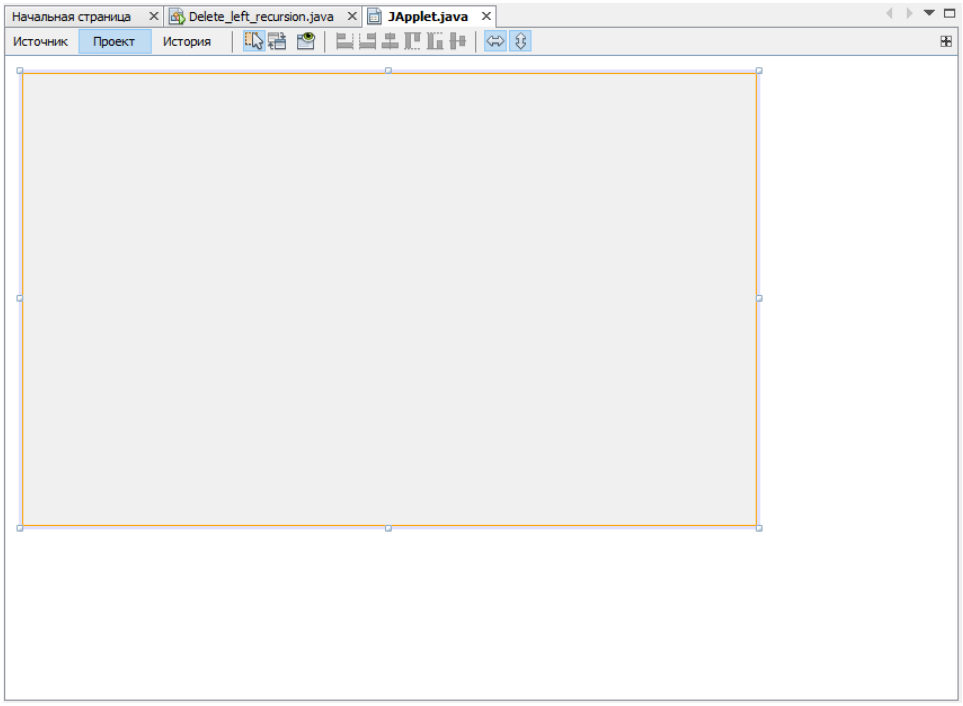

Рисунок 2.3 – Робоче вікно

Після цього потрібно встановити їй карткове розміщення для можливості подальшого перемикання між картками (рис. 2.4).

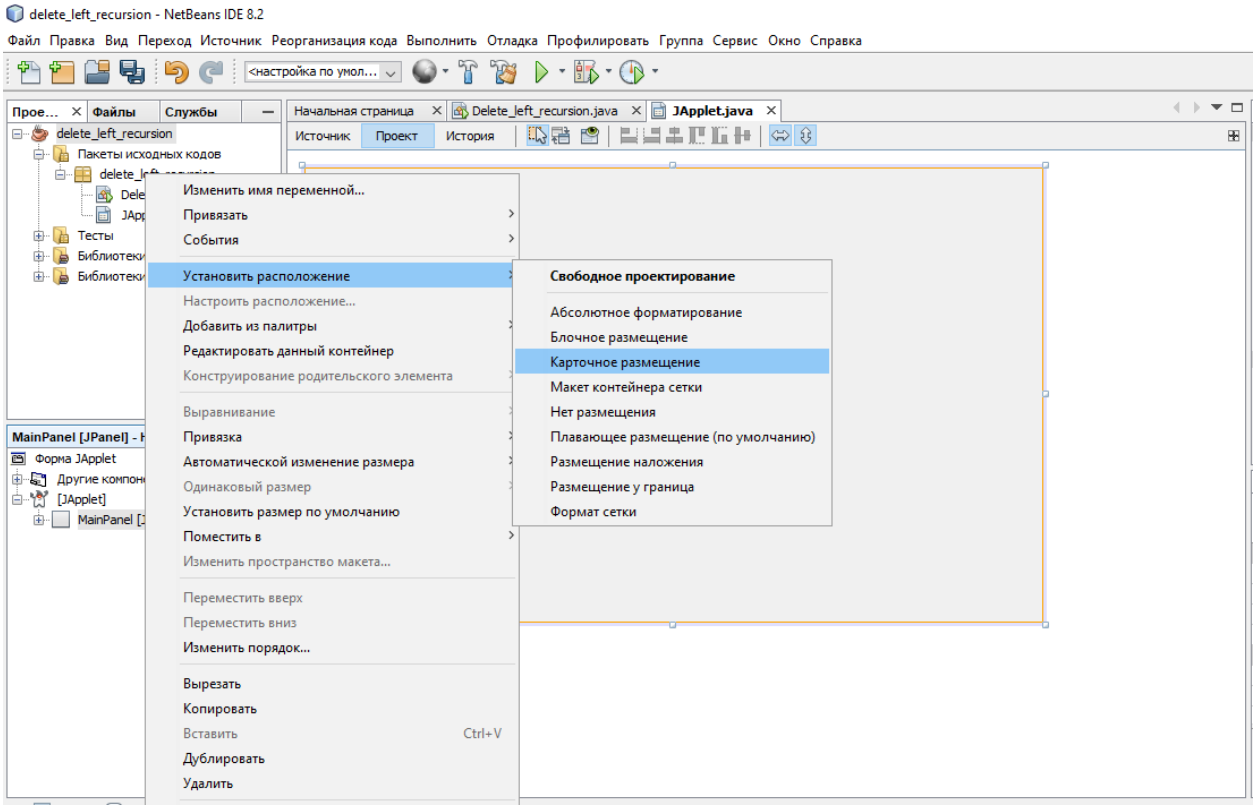

Рисунок 2.4 – Встановлення карткового розміщення

Після цього вже можна додавати елементи і розміщувати їх (рис. 2.5).

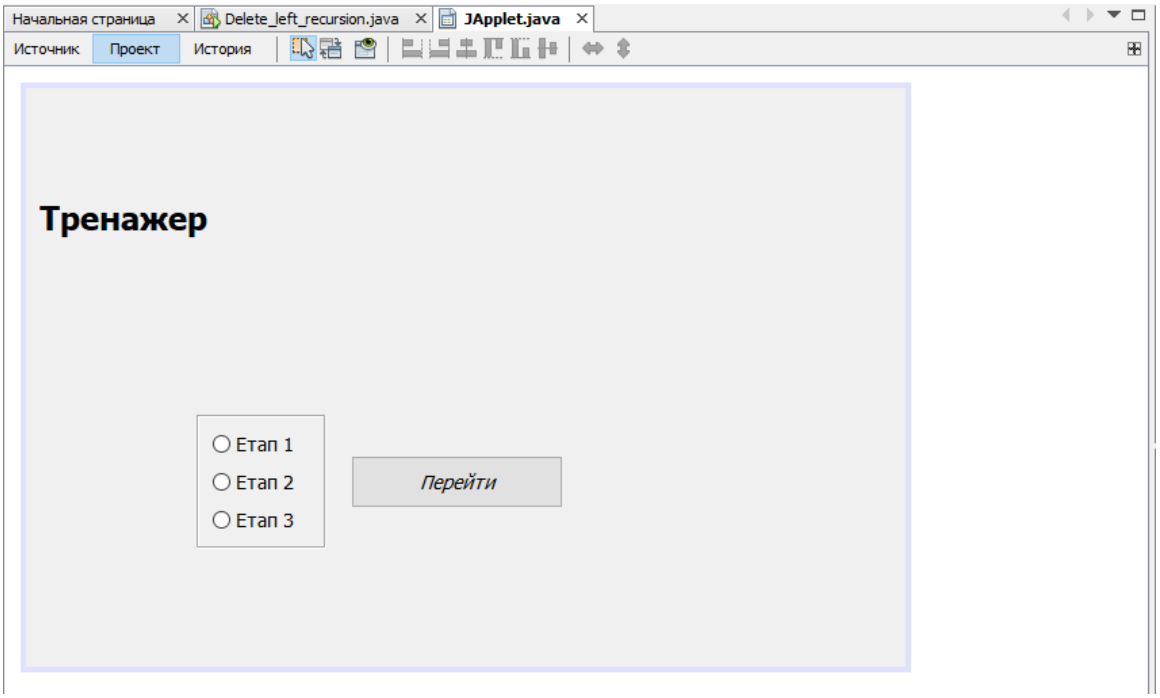

Рисунок 2.5 – Розміщення елементів

Для використання всіх можливостей було підключено бібліотеки:

```
import java.awt.CardLayout;
import java.awt.Desktop;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
```

Потім оголошено змінні step і cl:

```
    int step;
    CardLayout cl;
```

Для відкриття файлу з теоретичним матеріалом при натисканні на кнопку було створено подію TheoryActionPerformed.

```
    private void TheoryActionPerformed(java.awt.event.ActionEvent evt) {
        BufferedInputStream in = null;
        FileOutputStream fout = null;
        try {
            in = new
BufferedInputStream(getClass().getResource("theory.pdf").openStream());
            fout = new FileOutputStream("theory.pdf");
            byte data[] = new byte[1024];
            int count;
            while ((count = in.read(data, 0, 1024)) != -1) {
                fout.write(data, 0, count);
            }
        } catch (IOException ex) {
            Logger.getLogger(JApplet.class.getName()).log(Level.SEVERE,
null, ex);
        } finally {
            try {
                if ( in != null)
                    in.close();
                if (fout != null) {
                    fout.close();
                    if (Desktop.isDesktopSupported()) {
                        File file = new
File(System.getProperty("user.dir")+"\\theory.pdf");
                        Desktop.getDesktop().open(file);
                    }
                }
            }
            catch (IOException ex) {
                Logger.getLogger(JApplet.class.getName()).log(Level.SEVERE,
null, ex);
            }
        }
    }
```

Для кнопки «Перейти» розроблена StartActionPerformed.

```
    private void StartActionPerformed(java.awt.event.ActionEvent evt) {
        cl =(CardLayout) MainPanel.getLayout();
        cl.show(MainPanel, "card2");
        step=1;
    }
```

При натисненні кнопки «Наступний крок» спрацьовує подія NextActionPerformed. Якщо відкрите вікно з матеріалом, то просто відбудеться перехід по прикладу.

```
private void NextActionPerformed(java.awt.event.ActionEvent evt) {
    switch(step) {
        case 1:
            step++;
            Etap1Tab.setSelectedIndex(1);
            break;
        …
    }
}
```

В іншому разі відбувається перевірка відповіді. Якщо вона правильна, то розв’язування продовжується. Якщо ні – виводиться повідомлення про помилку.

```
        case 2:
            if("TE'".equals(jTextField1.getText())) {
                step++;
                Error1.setVisible(false);
                jTextField1.setEditable(false);
                jTextField2.setEnabled(true);
            } else {
                Error1.setVisible(true);
            }
            break;
        case 3:
            if("+TE'".equals(jTextField1.getText())) {
                step++;
                Error1.setVisible(false);
                jTextField1.setEditable(false);
                jTextField2.setEnabled(true);
            } else {
                Error1.setVisible(true);
            }
            break;
```

Також обов’язково при ініціалізації програми потрібно вказати, щоб текст помилки був прихований.

```
try {
    java.awt.EventQueue.invokeAndWait(new Runnable() {
        public void run() {
            initComponents();
            Error1.setVisible(false);
        }
    });
} catch (Exception ex) {
    ex.printStackTrace();
}
```

3.2. Інструкція по використанню програми

Після запуску відображається стартове вікно (рис. 2.6).

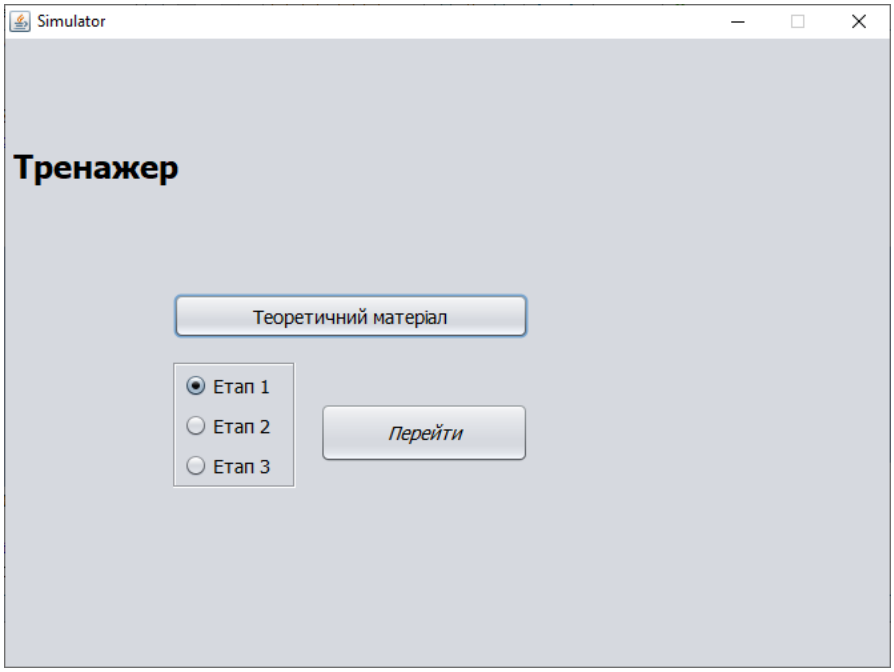

Рисунок 2.6 – Стартове вікно

Якщо натиснути «Теоретичний матеріал», то відкриється файл (рис. 2.7).

Після натиснення кнопки «Перейти» відобразиться матеріал для ознайомлення (рис. 2.8).

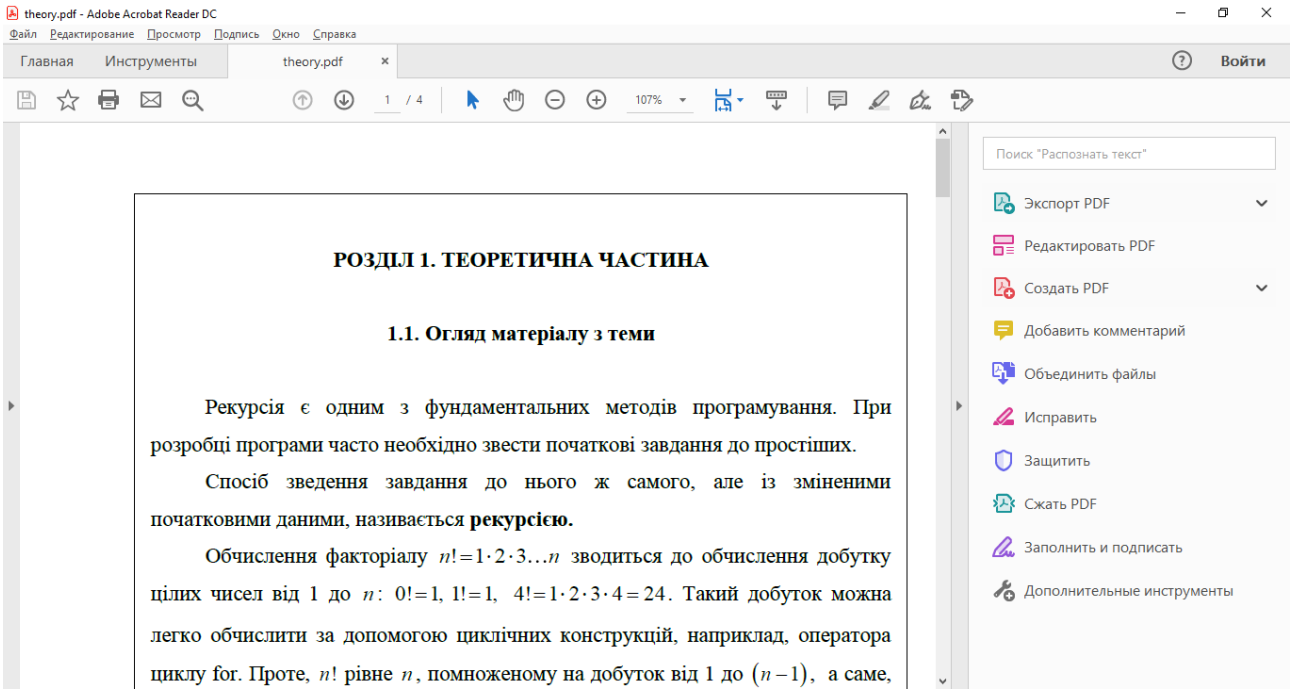

Рисунок 2.7 – Теоретичний матеріал

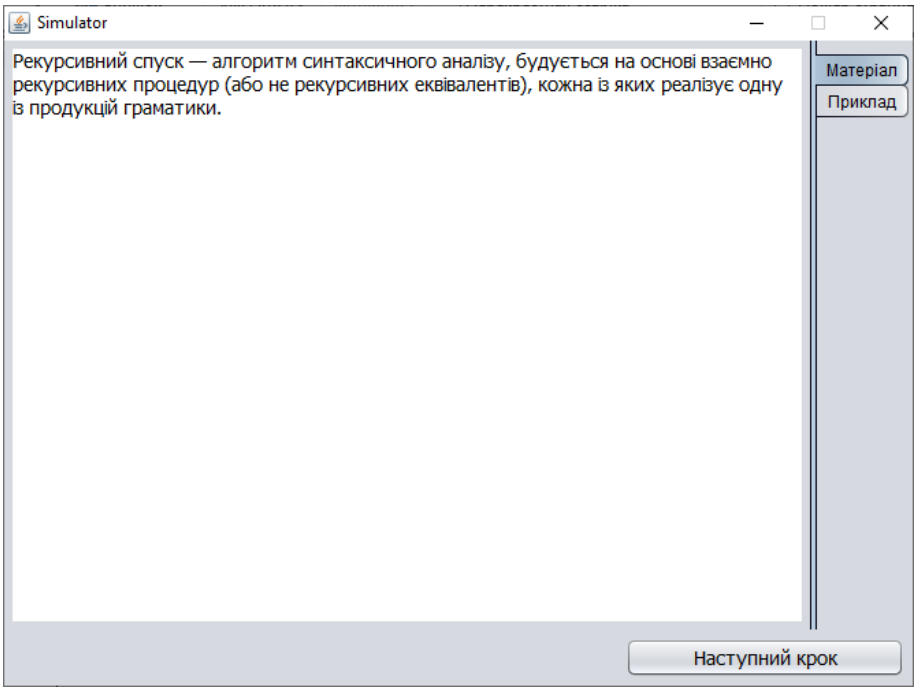

Рисунок 2.8 – Матеріал для ознайомлення

Для переходу потрібно натиснути «Наступний крок», перехід до розв’язання прикладу (рис. 2.9).

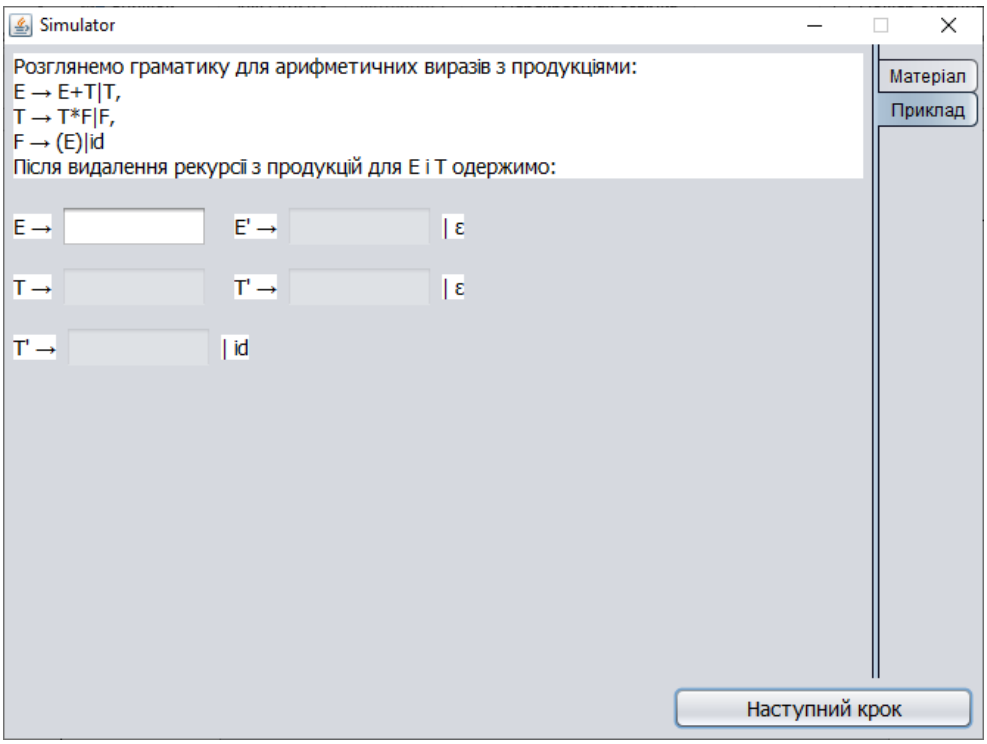

Рисунок 2.9 – Приклад

Якщо ввести неправильне значення відобразиться помилка (рис. 2.10).

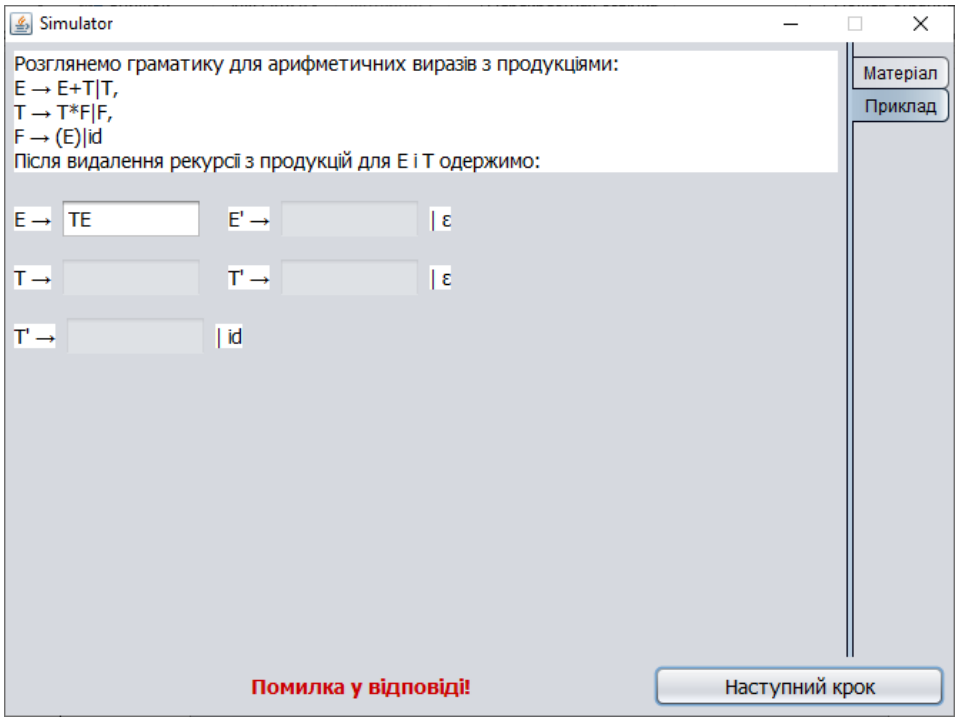

Рисунок 2.10 – Помилка

ВИСНОВКИ

Технологія дистанційного навчання – сукупність методів і засобів навчання та управління, що забезпечують проведення навчального процесу на відстані на основі використання сучасних інформаційних та телекомунікаційних технологій.

Варто підкреслити, що дистанційний курс, порівняно з традиційним навчанням, вимагає більшої гнучкості, детальнішої розробки змісту, ретельнішого планування, підтримки слухачів. Для створення власних дистанційних курсів викладач повинен уміти: визначити мету і завдання вивчення дистанційного курсу; визначити за допомогою тестування попередній рівень знань слухачів; поділити навчальну інформацію на окремі блоки; послідовно подати інформацію за певною логікою; складати запитання для закріплення змісту дистанційного курсу; розробити рекомендації з оформлення дистанційного курсу та окремих його частин; підтримувати мотивацію та зацікавленість слухачів у роботі з даним курсом.

Дистанційне навчання допомагає студентам і викладачам підвищувати ефективність навчального процесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ємець О. О. Методичні рекомендації щодо оформлення пояснювальних записок до курсових проектів (робіт) для студентів за освітньою програмою «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки та інформаційні технології», «Комп'ютерні науки» галузь знань - 12 «Інформаційні технології» / О. О. Ємець – Полтава : РВВ ПУЕТ, 2017. – 69с.
2. Дистанційний курс «Теорія програмування» // Головний центр дистанційного навчання вищого навчального закладу УКООПСПІЛКИ «Полтавський університет економіки і торгівлі»». – Режим доступу: http://www2.el.puet.edu.ua/st/course/view.php?id=2009
3. Бабій М.С. Теорія програмування: Навчальний посібник [Електронний ресурс] / М.С. Бабій, О.П. Чекалов.– Суми: Вид-во СумДУ, 2009. – 181 с.
4. Устранение левой рекурсии [Електронний ресурс] Викиконспекты. – Режим доступу: https://neerc.ifmo.ru/wiki/index.php?title=Устранение_левой_рекурсии
5. Кадомський К.К. Java. Теорія і практика: навчальний посібник для студентів природничих спеціальностей університетів [Текст] / К.К. Кадомський, П.К. Ніколюк / – Вінниця, Донну, 2019. – 197 с.
6. 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