МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

ОСНОВИ АЛГОРИТМІЗАЦІЇ

Методичні рекомендації до виконання лабораторних робіт

для студентів спеціальності 121 "Інженерія програмного забезпечення" освітньої програми "Інженерія програмного забезпечення" першого (бакалаврського) рівня

Харків
ХНЕУ ім. С. Кузнеця
2023

УДК 004.421(072.034)
О-75

Укладачі: О. В. Щербаков
О. В. Фролов

Затверджено на засіданні кафедри інформаційних систем.
Протокол № 8 від 21.11.2022 р.

Самостійне електронне текстове мережеве видання

Основи алгоритмізації [Електронний ресурс] : методичні ре-
О-75 комендації до виконання лабораторних робіт для студентів спеціальності 121 "Інженерія програмного забезпечення" освітньої програми "Інженерія програмного забезпечення" першого (бакалаврського) рівня / уклад. О. В. Щербаков, О. В. Фролов. – Харків : ХНЕУ ім. С. Кузнеця, 2023. – 68 с.

Подано методичні рекомендації до виконання лабораторних робіт, метою яких є закріплення та поглиблення знань теоретичного матеріалу, набуття навичок розроблення алгоритмів. За кожною лабораторною роботою визначено мету, завдання, засоби та порядок виконання, зміст звіту та контрольні запитання.

Рекомендовано для студентів спеціальності 121 "Інженерія програмного забезпечення" освітньої програми "Інженерія програмного забезпечення" першого (бакалаврського) рівня.

УДК 004.421(072.034)

Вступ

Широке розповсюдження інформаційних технологій, науково-технічний прогрес, проникнення інформаційно-комунікаційних технологій в усі сфери людської діяльності висувають нові, підвищені вимоги до підготовки фахівців в галузі інформаційних технологій.

Сучасний професіонал у цій галузі повинен володіти цілим багатьма компетентностями, серед яких особливе місце відводять загальнонауковим та загальнотехнічним компетентностям, тобто – фундаментальним знанням.

У загальному навчальна дисципліна "Основи алгоритмізації" розглядає такі питання, як формалізація понять "алгоритм", "складність алгоритму" та дослідження формальних алгоритмічних систем; загальні принципи побудови ефективних алгоритмів; сучасні методи дослідження та аналізу алгоритмів; способи та механізми реалізації ефективних алгоритмів у конкретних застосуваннях; класифікація завдань, визначення і дослідження класів складності; асимптотичний аналіз складності алгоритмів; дослідження та аналіз рекурсивних алгоритмів; отримання явних функцій трудомісткості для порівняльного аналізу алгоритмів; розроблення критеріїв порівняльного оцінювання якості алгоритмів.

Мета навчальної дисципліни: отримання студентами ґрунтовної математичної підготовки та знань теоретичних, методичних і алгоритмічних основ інформаційних технологій для їх використання під час вирішення прикладних і наукових завдань у сфері інформаційних систем і технологій, забезпечення теоретичної та інженерної підготовки фахівців у галузі проєктування, впровадження та використання інформаційних систем у бізнесі. Ознайомити студентів з сучасними та ефективними алгоритмами комп'ютерного оброблення інформації, а також методами їх дослідження та аналізу.

Навчальна дисципліна "Основи алгоритмізації" відіграє важливу роль у підготовці студентів за спеціальністю 121 "Інженерія програмного забезпечення". Відповідно до освітньо-професійної програми підготовки бакалаврів з інженерії програмного забезпечення навчальна дисципліна бере участь у формуванні та розвитку таких компетентностей:

- здатність до алгоритмічного та логічного мислення;
- здатність до абстрактного мислення, аналізу та синтезу;

- здатність застосовувати знання у практичних ситуаціях;
- здатність ідентифікувати, класифікувати та формулювати вимоги до програмного забезпечення;
- здатність брати участь у проєктуванні програмного забезпечення, включаючи проведення моделювання (формальний опис) його структури, поведінки та процесів функціонування;
- здатність аналізувати, вибирати і застосовувати методи та засоби для забезпечення інформаційної безпеки (у тому числі кібербезпеки).

У результаті вивчення навчальної дисципліни "Основи алгоритмізації" студенти набувають таких результатів у навчанні:

- застосовувати на практиці ефективні підходи щодо проєктування програмного забезпечення;
- знати і застосовувати методи розроблення алгоритмів, конструювання програмного забезпечення та структур даних і знань.

Методичні рекомендації до виконання лабораторних робіт складені відповідно до затвердженої робочої програми з навчальної дисципліни та охоплюють практичні питання щодо розроблення алгоритмів за основними темами цієї програми.

Лабораторний практикум охоплює шість лабораторних робіт. За результатами виконання кожної роботи студент має оформити електронний звіт засобами *Word*. З кожного завдання потрібно записати формулювання завдання, подати опис алгоритму графічними засобами та його словесний опис (або опис за допомогою псевдокоду), а також навести тестові приклади з результатами виконання. У висновках з лабораторної роботи слід дати самооцінку тому, які нові знання, уміння та навички отримано під час виконання роботи.

Лабораторна робота 1
Розроблення схем алгоритмів лінійних обчислювальних процесів

Мета лабораторної роботи: набути практичних навичок розроблення схем лінійних обчислювальних процесів.

Теоретична частина

Алгоритм – це заздалегідь визначена точна послідовність дій, яка задає дискретний (поетапний) процес, що починається певним чином й приводить до отримання результату за кінцеве число етапів. Також алгоритм може бути визначеним, як задана послідовність етапів, виконання якої гарантовано приведе до отримання очікуваного результату.

Найбільш поширеним способом подання алгоритму є графічний. У графічному поданні алгоритми зображають у вигляді схеми, доповненої елементами словесного або математичного запису.

Схема алгоритму містить геометричні фігури (блокові символи), з'єднані між собою стрілками (лініями), що вказують порядок виконання операцій.

Блокові символи стандартизовані і визначені в ГОСТ 19.701-90 "Схеми алгоритмів, програм, даних і систем. Умовніпозначення і правила виконання".

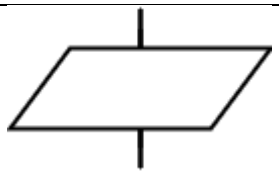
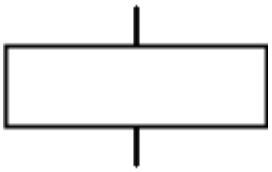
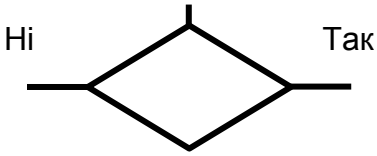
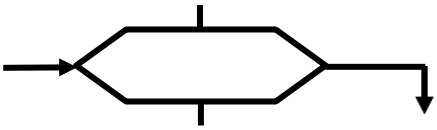
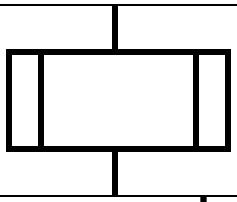
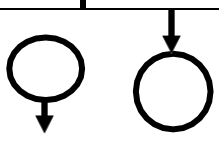
Єдине обмеження накладають на послідовність записів – їх слід читати зліва направо і зверху вниз незалежно від напрямку потоків інформації.

Логіка алгоритму повинна спиратися на мінімальну кількість досить простих базових структур. У процесі розроблення схем алгоритмів необхідно дотримуватися деяких вимог:

1. У схемі алгоритму всі лінії від блоку "початок" до блоку "кінець" не повинні мати розривів, які не позначені з'єднаннями. Усі лінії, що вказуютьпослідовність виконання дій, повинні бути замкненими (табл. 1.1).

Таблиця 1.1

Види і значення основних алгоритмічних блоків

№ з/п	Зображення	Значення
1		Блоки початку і кінця програми
2		Блок введення або виведення інформації
3		Блок обчислень
4		Логічний блок
5		Блок модифікації(початок циклу)
6		Блок підпрограми
7		Внутрішні сторінкові сполучні блоки
8		Міжсторінкові сполучні блоки
9		Блок коментарів

У схемі повинні чітко простежуватися потоки інформації. Блоки слід розміщувати таким чином, щоб уникати перетину ліній. Під час передавання управління в схемі "знизу-вгору" або "справа-наліво" лінії обов'язково позначають стрілками.

2. Не допускають передавання управління в нікуди. "Джерело" передавання управління і "одержувач" повинні бути чітко позначені.

Лінійні алгоритми

Лінійними називають алгоритми, в яких операції виконують послідовно одна за одною, в природному і єдиному порядку проходження. У таких алгоритмах усі блоки мають послідовне з'єднання логічним зв'язком передавання інформаційних потоків. У них можуть бути використані всі блоки, за винятком блоків перевірки умови і модифікації. Лінійні алгоритми, як правило, є складовою частиною будь-якого алгоритмічного процесу.

Для подання такого алгоритму використовують алгоритмічну конструкцію *слідування* (послідовне виконання), яка передбачає послідовне виконання дій у тому порядку, в якому вони записані в тексті програми. Подання цієї конструкції у блок-схемах здійснюють послідовністю блоків "процес" (рис. 1.1).

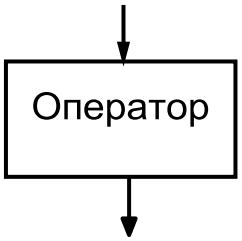


Рис. 1.1. **Конструкція слідування**

Типовим прикладом лінійного алгоритму є процедура обчислення за певними формулами.

Наприклад, на рис. 1.2 зображено схему лінійного алгоритму для обчислення суми двох чисел.

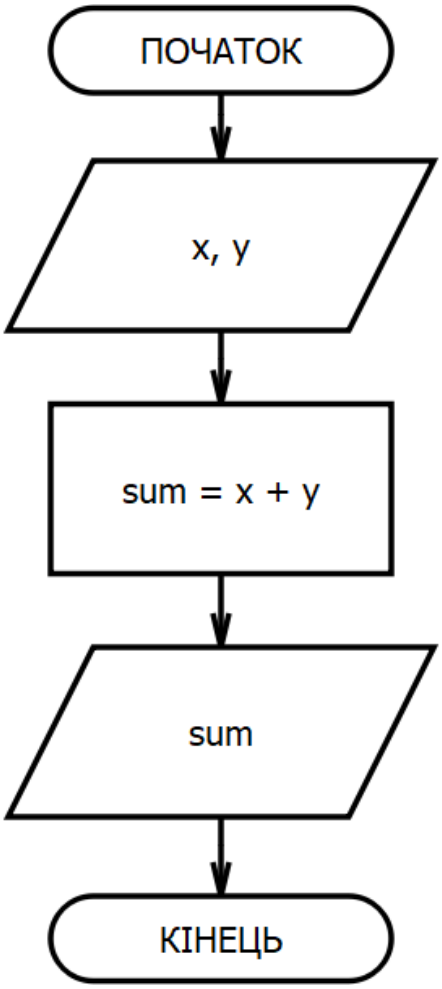


Рис. 1.2. **Лінійний алгоритм обчислення суми двох чисел**

Практична частина

Розроблення лінійних алгоритмів

Система оцінювання. За правильно виконане перше завдання студент отримує 4 бали; за правильно виконані перше та друге завдання – 6 балів; якщо правильно виконані всі три завдання – 7 балів.

Приклад. Розробити алгоритм обчислення площі паралелограма за заданою стороною *a* та висотою *h* (рис. 1.3).

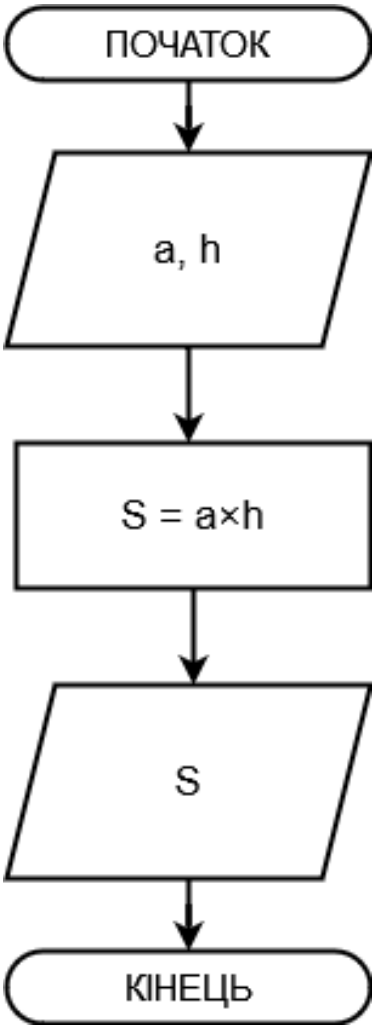


Рис. 1.3. **Блок-схема алгоритму**

Завдання 1. Розробити алгоритм для вирішення такого завдання (за варіантами):

1. Дано сторону квадрата *a*. Знайти його периметр $P = 4 \times a$.

2. Дано сторону квадрата *a*. Знайти його площу $S = a^2$.

3. Дано сторони прямокутника *a* і *b*. Знайти його площу $S = a \times b$.

4. Дано сторони прямокутника *a* і *b*. Знайти його периметр $P = 2 \times (a + b)$.

5. Дано діаметр кола *d*. Знайти його довжину $L = \pi \times d$. Як значення π використовувати 3.14.

6. Дано довжину ребра куба *a*. Знайти об'єм куба $V = a^3$.

7. Дано довжину ребра куба a. Знайти площу його поверхні $S = 6 \times a^2$.

8. Дано довжини ребер *a*, *b*, *c* прямокутного паралелепіпеда. Знайти його об'єм $V = a \times b \times c$.

9. Дано довжини ребер *a*, *b*, с прямокутного паралелепіпеда. Знайти площу його поверхні $S = 2 \times (a \times b + b \times c + a \times c)$.

10. Знайти довжину кола *L* заданого радіуса *r.* L = $2 \times \pi \times$ r. Як значення π використовувати 3.14.

11. Знайти площу круга *S* заданого радіуса *r.* $S = \pi \times r^2$. Як значення π використовувати 3.14.

12. Дано катети прямокутного трикутника *a* і *b*. Знайти його площу $S = (a \times b) / 2$.

13. Дано довжини сторін трикутника *a*, *b*, *c*. Знайти його периметр $P = a + b + c$.

14. Дано сторону трикутника *a* і висоту *h*. Знайти його площу $S = (a \times h) / 2$.

15. Знайти об'єм кулі заданого радіуса *r*: $V = 4/3 \times \pi \times r^3$. Як значення π використовувати 3.14.

16. Знайти площу поверхні кулі заданого радіуса *r*: $S = 4 \times \pi \times r^2$. Як значення π використовувати 3.14.

17. Дано сторону квадрата *a*. Знайти його периметр $P = 4 \times a$.

18. Дано сторону квадрата *a*. Знайти його площу $S = a^2$.

19. Дано сторони прямокутника *a* і *b*. Знайти його площу $S = a \times b$.

20. Дано сторони прямокутника *a* і *b*. Знайти його периметр $P = 2 \times (a + b)$.

21. Дано діаметр кола *d*. Знайти його довжину $L = \pi \times d$. Як значення π використовувати 3.14.

22. Дано довжину ребра куба *a*. Знайти об'єм куба $V = a^3$.

23. Дано довжину ребра куба *a*. Знайти площу його поверхні $S = 6 \times a^2$.

24. Дано довжини ребер *a*, *b*, *c* прямокутного паралелепіпеда. Знайти його об'єм $V = a \times b \times c$

25. Дано довжини ребер a, b, с прямокутного паралелепіпеда. Знайти площу його поверхні $S = 2 \times (a \times b + b \times c + a \times c)$.

Завдання 2. Розробити алгоритм для вирішення такого завдання (заваріантами):

1. Дано тризначне додатне ціле число *x*. Знайти суму його першої та останньої цифр.

2. Дано тризначне додатне ціле число *x*. Знайти суму його другої та останньої цифр.

3. Дано тризначне додатне ціле число *x*. Знайти суму його першої та другої цифр.

4. Дано тризначне додатне ціле число *x*. Знайти суму всіх його цифр.

5. Дано чотиризначне додатне ціле число *x*. Знайти суму його першої та останньої цифр.

6. Дано чотиризначне додатне ціле число *x*. Знайти суму його першої та другої цифр.

7. Дано чотиризначне додатне ціле число *x*. Знайти суму його першої та третьої цифр.

8. Дано чотиризначне додатне ціле число *x*. Знайти суму його другої та останньої цифр.

9. Дано чотиризначне додатне ціле число *x*. Знайти суму його третьої та останньої цифр.

10. Дано чотиризначне додатне ціле число *x*. Знайти суму його другої та третьої цифр.

11. Дано чотиризначне додатне ціле число *x*. Знайти суму всіх його цифр.

12. Дано тризначне додатне ціле число *x*. Знайти суму його першої таостанньої цифр.

13. Дано тризначне додатне ціле число *x*. Знайти різницю його другої та останньої цифр.

14. Дано тризначне додатне ціле число *x*. Знайти різницю його першої та другої цифр.

15. Дано тризначне додатне ціле число *x*. Знайти добуток усіх його цифр.

16. Дано чотиризначне додатне ціле число *x*. Знайти різницю його першої та останньої цифр.

17. Дано чотиризначне додатне ціле число *x*. Знайти різницю його першої та другої цифр.

18. Дано чотиризначне додатне ціле число *x*. Знайти різницю його першої та третьої цифр.

19. Дано чотиризначне додатне ціле число *x*. Знайти різницю його другої та останньої цифр.

20. Дано чотиризначне додатне ціле число *x*. Знайти різницю його третьої та останньої цифр.

21. Дано чотиризначне додатне ціле число *x*. Знайти різницю його другої та третьої цифр.

22. Дано чотиризначне додатне ціле число *x*. Знайти добуток всіх його цифр.

23. Дано тризначне додатне ціле число *x*. Знайти суму його першої та останньої цифр.

24. Дано тризначне додатне ціле число *x*. Знайти суму його другої та останньої цифр.

25. Дано тризначне додатне ціле число *x*. Знайти суму його першої та другої цифр.

Завдання 3. Розробити алгоритм для вирішення такого завдання (спільний варіант).

Дано коло, довжина якого дорівнює *L*. Навколо кола описали один квадрат. У коло вписали другий квадрат (рис. 1.4). Задано значення *L*. Знайти площу першого квадрата та периметр другого.

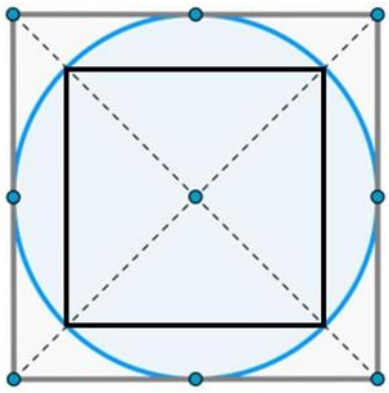


Рис. 1.4. **Зображення фігур до завдання 3**

Контрольні запитання до лабораторної роботи 1

1. Що таке алгоритм?
2. У чому полягає властивість зрозумілості алгоритму?
3. У чому полягає властивість масовості алгоритму?
4. Який блок під час запису блок-схеми має дві лінії виходу?
5. Які алгоритми належать до лінійних?

Лабораторна робота 2
Розроблення схем алгоритмів обчислювальних процесів, що розгалужуються

Мета роботи: набути практичних навичок розроблення схем обчислювальних процесів, що розгалужуються.

Теоретична частина

Алгоритми з розгалуженням

Під час складання схем алгоритмів часто виникає необхідність проведення аналізу вихідних даних або проміжних результатів обчислень і визначення подальшого порядку виконання обчислювального процесу залежно від результатів цього аналізу. Алгоритми, в яких залежно від виконання деякої логічної умови відбувається розгалуження обчислень за одним із декількох можливих напрямків, називають алгоритми, що розгалужуються, або алгоритми з розгалуженням. Подібні алгоритми передбачають вибір одного з альтернативних шляхів продовження обчислень. Кожний можливий напрямок обчислень називають гілкою. Логічну умову називають простою, якщо процес, що розгалужується, має дві гілки, і складною, якщо процес розгалужується на три і більше гілки.

Структура розгалуження існує в чотирьох основних варіантах:

- "якщо-то-інакше" (повне розгалуження);
- "якщо-то" (неповне розгалуження);
- "вибір-інакше" (повний багатоваріантний вибір);
- "вибір" (неповний багатоваріантний вибір).

Повне розгалуження. Ця алгоритмічна структура передбачає виконання певних дій як у разі виконання, так і у разі невиконання заданої умови. Графічне подання такої структури у блок-схемах має вигляд (рис. 2.1):

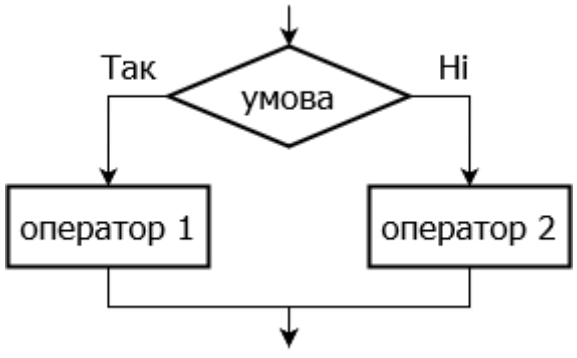
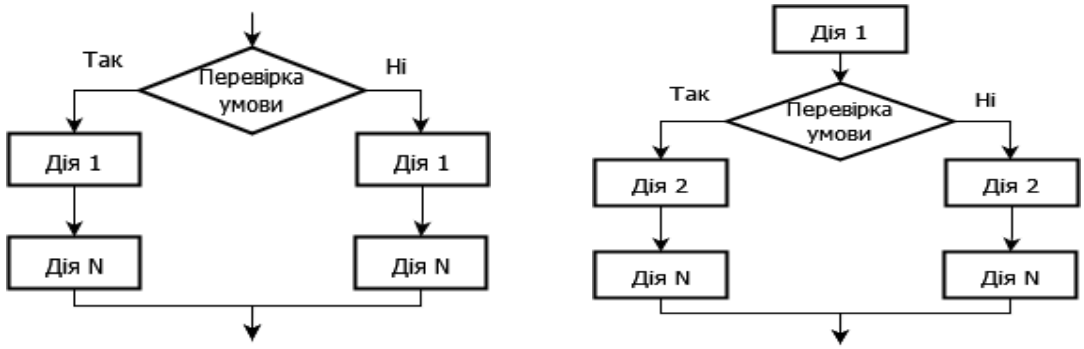


Рис. 2.1. **Повне розгалуження**

При цьому умову формулюють таким чином, щоб відповідь перевірки була лише "так" чи "ні" (рис. 2.2а, б). Наприклад:



а) б)

Рис. 2.2. **Блок-схеми повного розгалуження**

Неповне розгалуження. Ця алгоритмічна структура передбачає виконання дій тільки у разі виконання, або у разі невиконання заданої умови. Тобто одна із її гілок взагалі не передбачає ніяких дій. Графічне подання таких структур у блок-схемах має вигляд (рис. 2.3):

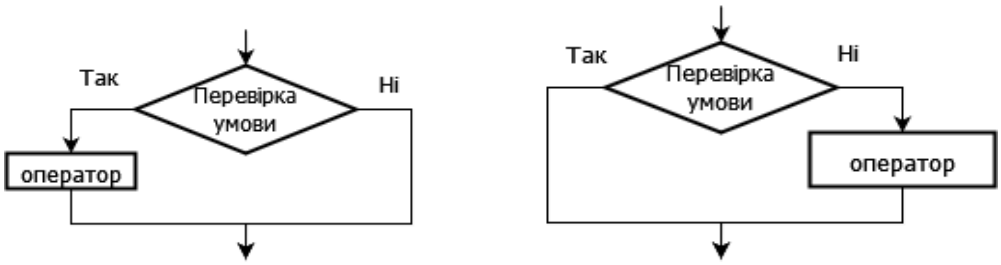


Рис. 2.3. **Неповне розгалуження**

Залежно від того, на скільки гілок розгалужується алгоритм, він може бути простим або складним. Для простого розгалуженого процесу перевіряють одну умову, для складного – дві чи більше умов, кожна з яких відокремлює одну гілку (рис. 2.4).

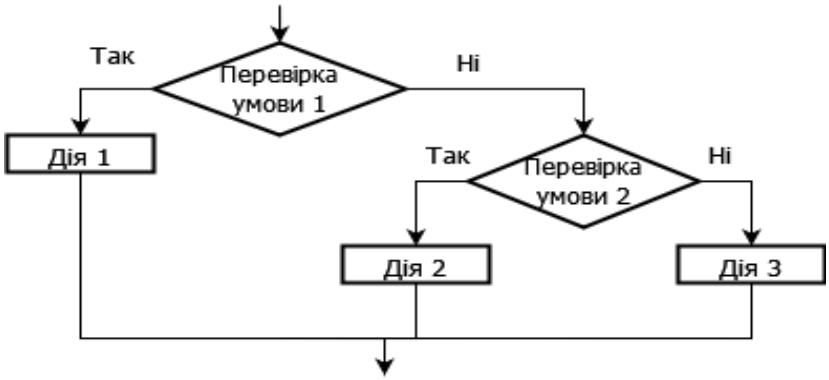
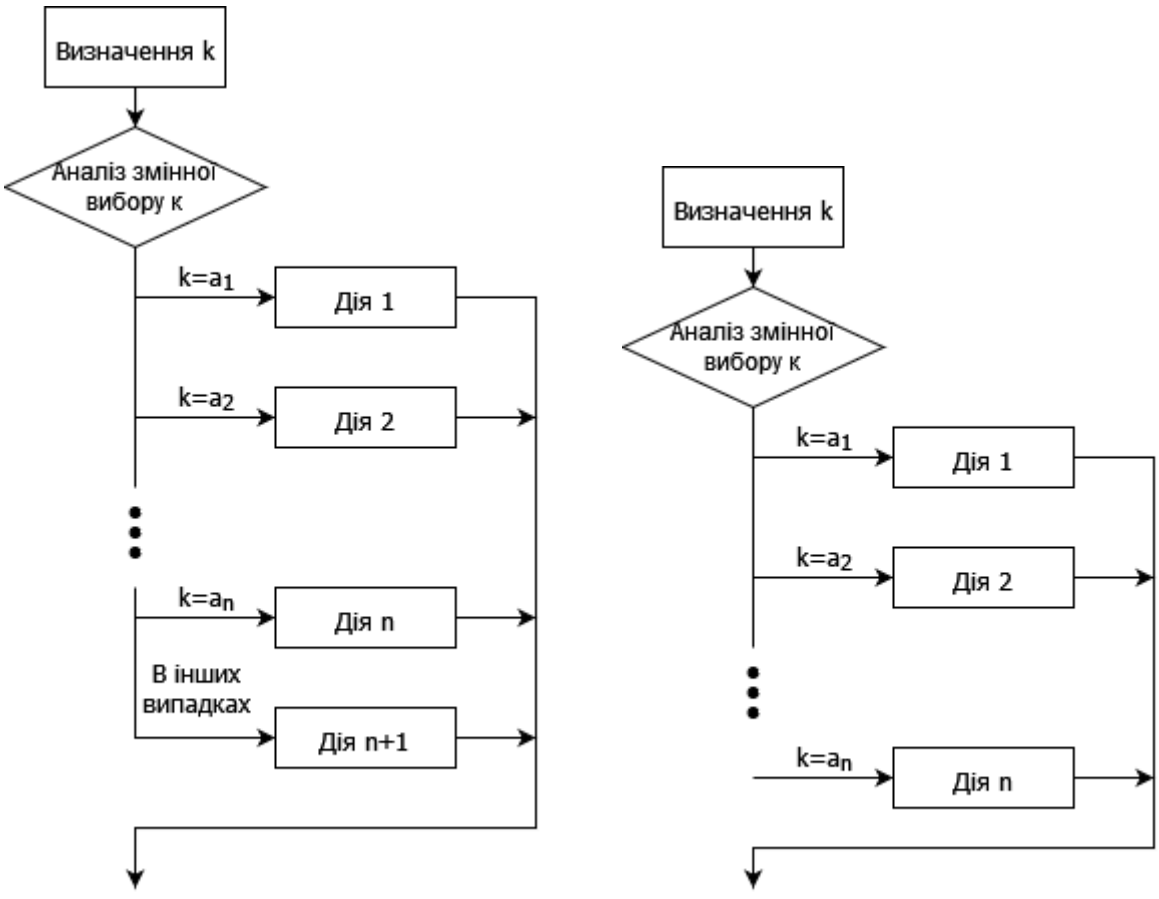


Рис. 2.4. **Блок-схема складного розгалуження**

Багатоваріантний вибір. Збільшення кількості умов робить алгоритм більш заплутаним, він втрачає наочність, перевірити його правильність досить складно. У таких випадках необхідно перехід до будь-якої гілки розгалуженого алгоритму пов'язати з деякою змінною, кожне значення якої відповідатиме одній із гілок розгалуження, тобто одному з варіантів оброблення інформації. Це можуть бути не тільки окремі значення, а й проміжки значень, до яких належатиме конкретне значення змінної. Тоді всі логічні блоки алгоритму об'єднують в один блок аналізу цієї змінної, який матиме не два виходи, а стільки, скільки існує варіантів оброблення. Таке розгалуження називають *багатоваріантним*.

Якщо для такого розгалуження передбачений варіант оброблення, коли значення змінної не належить до перелічених значень, то таке розгалуження є повним багатоваріантним вибором (рис. 2.5а), інакше – неповним багатоваріантним вибором (рис. 2.5б).



а) повний б) неповний

Рис. 2.5. **Блок-схеми багатоваріантного вибору**

Практична частина

Система оцінювання. За правильно виконане перше завдання студент отримує 4 бали; за правильно виконані перше та друге завдання – 6 балів; якщо правильно виконано всі три завдання – 7 балів.

Приклад. Розробити алгоритм для вирішення такого завдання: дано довільне ціле число *x*. Якщо воно більше 10, то додати до нього 25, а якщо ні, то відняти від нього 15 (рис. 2.6).

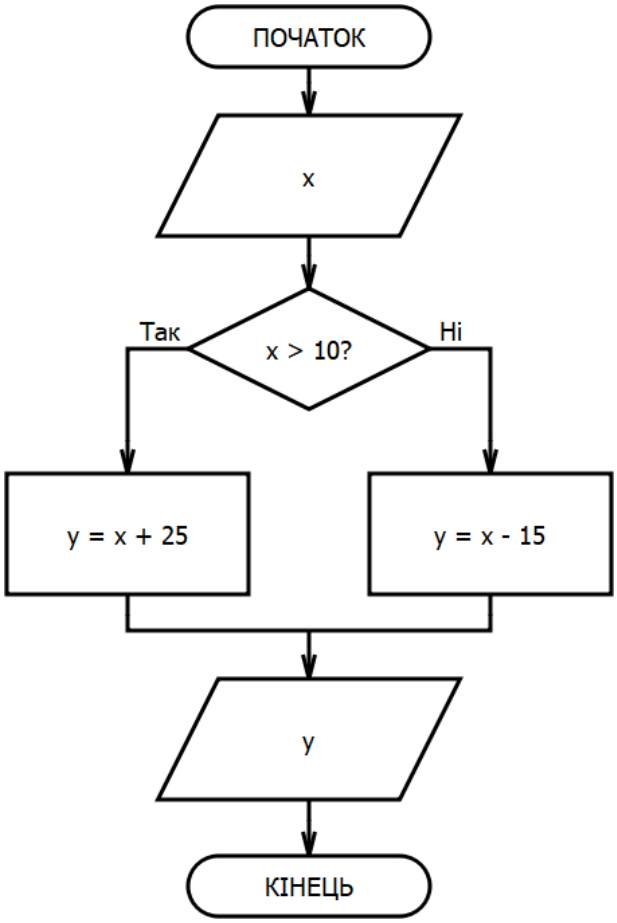


Рис. 2.6. **Блок-схема алгоритму**

Завдання 1. Розробити алгоритм для вирішення завдання (за варіантами):

1. Дано довільне ціле число *x*. Якщо воно парне, то поділити його на 2, а якщо ні, то помножити його на 3.

2. Дано довільне ціле число *x*. Якщо воно додатне, то помножити його на 5, а якщо ні, то додати до нього 15.

3. Дано довільне ціле число *x*. Якщо воно від'ємне, то додати до нього 7, а якщо ні, то відняти від нього 10.

4. Дано довільне ціле число *x*. Якщо воно більше 4, то додати до нього 11, а якщо ні, то помножити його на 4.

5. Дано довільне ціле число *x*. Якщо воно менше 20, то додати до нього 3, а якщо ні, то відняти від нього 5.

6. Дано довільне ціле число *x*. Якщо воно непарне, то додати до нього 1, а якщо ні, то поділити його на 2.

7. Дано довільне ціле число *x*. Якщо воно парне, то додати до нього 5, а якщо ні, то відняти від нього 1.

8. Дано довільне ціле число *x*. Якщо воно додатне, то відняти до нього 12, а якщо ні, то помножити його на 10.

9. Дано довільне ціле число *x*. Якщо воно від'ємне, то помножити його на 6, а якщо ні, то додати до нього 8.

10. Дано довільне ціле число *x*. Якщо воно більше 3, то додати до нього 17, а якщо ні, то відняти від нього 2.

11. Дано довільне ціле число *x*. Якщо воно менше 9, то відняти від нього 5, а якщо ні, то помножити його на 3.

12. Дано довільне ціле число *x*. Якщо воно кратне 3, то поділити його на 3, а якщо ні, то додати до нього 1.

13. Дано довільне ціле число *x*. Якщо воно більше 7, то додати до нього 11, а якщо ні, то помножити його на 4.

14. Дано довільне ціле число *x*. Якщо воно парне, то поділити його на 2, а якщо ні, то помножити його на 3.

15. Дано довільне ціле число *x*. Якщо воно додатне, то помножити його на 5, а якщо ні, то додати до нього 15.

16. Дано довільне ціле число x. Якщо воно від'ємне, то додати до нього 7, а якщо ні, то відняти від нього 10.

17. Дано довільне ціле число *x*. Якщо воно більше 4, то додати до нього 11, а якщо ні, то помножити його на 4.

18. Дано довільне ціле число *x*. Якщо воно менше 20, то додати до нього 3, а якщо ні, то відняти від нього 5.

19. Дано довільне ціле число *x*. Якщо воно непарне, то додати до нього 1, а якщо ні, то поділити його на 2.

20. Дано довільне ціле число *x*. Якщо воно парне, то додати до нього 5, а якщо ні, то відняти від нього 1.

21. Дано довільне ціле число *x*. Якщо воно додатне, то відняти до нього 12, а якщо ні, то помножити його на 10.

22. Дано довільне ціле число *x*. Якщо воно від'ємне, то помножити його на 6,а якщо ні, то додати до нього 8.

23. Дано довільне ціле число *x*. Якщо воно більше 3, то додати до нього 17, а якщо ні, то відняти від нього 2.

24. Дано довільне ціле число *x*. Якщо воно менше 9, то відняти від нього 5, а якщо ні, то помножити його на 3.

25. Дано довільне ціле число *x*. Якщо воно кратне 3, то поділити його на 3, а якщо ні, то додати до нього 1.

Завдання 2. Розробити алгоритм для вирішення завдання (за варіантами):

1. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* більше, ніж *y*, то у випадку, якщо *x* – парне число, то поділити його на 2, а якщо ні – помножити на 3. Якщо значення *x* не більше, ніж *y*, то додати до у число 5.

2. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* менше, ніж *y*, то *y* випадку, якщо *x* – додатне число, то помножити його на 2, а якщо ні – відняти від нього 7. Якщо значення *x* не менше, ніж *y*, то відняти від *y* число 1.

3. Дано два довільних цілих числа *x* та *y*. Якщо значення суми *x* та *y* більше 10, то у випадку, якщо *x* – непарне число, то помножити його на 4, а якщо ні – поділити його на 2. Якщо значення суми *x* та *y*не більше 10, то помножити *y* на число 9.

4. Дано два довільних цілих числа *x* та *y*. Якщо значення x більше, ніж *y*, то у випадку, якщо *x* – непарне число, то додати до нього 1, а якщо ні – поділити його на 2. Якщо значення *x* не більше, ніж *y*, то помножити *y* на число 4.

5. Дано два довільних цілих числа *x* та *y*. Якщо значення x менше, ніж*y*, то у випадку, якщо *x* – від'ємне число, то обчислити його модуль, а якщо ні – додати нього 12. Якщо значення *x* не менше, ніж *y*, то помножити у на число 3.

6. Дано два довільних цілих числа *x* та *y*. Якщо значення суми *x* та *y* менше 24, то у випадку, якщо *x* – непарне число, то додати до нього число 13, а якщо ні – поділити його на 2. Якщо значення суми *x* та *y* не менше 24, то поділити у на число 4.

7. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* більше, ніж *y*, то у випадку, якщо *x* – парне число, то поділити його на 2, а якщо ні – помножити на 5. Якщо значення *x* не більше, ніж *y*, то додати до *y* число 7.

8. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* менше, ніж *y*, то у випадку, якщо *x* – додатне число, то помножити його на 4, а якщо ні – відняти від нього 8. Якщо значення *x* не менше, ніж *y*, то відняти від *y* число 3.

9. Дано два довільних цілих числа *x* та *y*. Якщо значення суми *x* та *y* більше 15, то у випадку, якщо *x* – непарне число, то помножити його на 3, а якщо ні – поділити його на 2. Якщо значення суми *x* та *y* не більше 15, то помножити *y* на число 11.

10. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* більше, ніж *y*, то у випадку, якщо *x* – непарне число, то додати до нього 9, а якщо ні – поділити його на 2. Якщо значення *x* не більше, ніж *y*, то помножити *y* на число 12.

11. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* менше, ніж *y*, то у випадку, якщо *x* – від'ємне число, то обчислити його модуль, а якщо ні – додати нього 32. Якщо значення *x* не менше, ніж *y*, то помножити *y* на число 6.

12. Дано два довільних цілих числа *x* та *y*. Якщо значення суми *x* та *y* менше 18, то у випадку, якщо *x* – непарне число, то додати до нього число 13, а якщо ні – поділити його на 2. Якщо значення суми *x* та *y* не менше 18, то поділити *y* на число 4.

13. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* більше, ніж *y*, то у випадку, якщо *x* – парне число, то поділити його на 2, а якщо ні – помножити на 3. Якщо значення *x* не більше, ніж *y*, то додати до *y* число 5.

14. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* менше, ніж *y*, то у випадку, якщо *x* – додатне число, то помножити його на 2, а якщо ні – відняти від нього 7. Якщо значення *x* не менше, ніж *y*, то віднятивід *y* число 1.

15. Дано два довільних цілих числа *x* та *y*. Якщо значення суми *x* та *y* більше 10, то у випадку, якщо *x* – непарне число, то помножити його на 4, а якщо ні – поділити його на 2. Якщо значення суми *x* та *y* не більше 10, то помножити *y* на число 9.

16. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* більше, ніж *y*, то у випадку, якщо *x* – непарне число, то додати до нього 1, а якщо ні – поділити його на 2. Якщо значення *x* не більше, ніж *y*, то помножити *y* на число 4.

17. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* менше, ніж *y*, то у випадку, якщо *x* – від'ємне число, то обчислити його модуль, а якщо ні – додати нього 12. Якщо значення *x* не менше, ніж *y*, то помножити *y* на число 3.

18. Дано два довільних цілих числа *x* та *y*. Якщо значення суми *x* та *y* менше 24, то у випадку, якщо *x* – непарне число, то додати до нього число 13, а якщо ні – поділити його на 2. Якщо значення суми *x* та *y* не менше 24, то поділити *y* на число 4.

19. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* більше, ніж *y*, то у випадку, якщо *x* – парне число, то поділити його на 2, а якщо ні – помножити на 3. Якщо значення *x* не більше, ніж *y*, то додати до *y* число 5.

20. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* менше, ніж *y*, то у випадку, якщо *x* – додатне число, то помножити його на 2, а якщо ні – відняти від нього 7. Якщо значення *x* не менше, ніж *y*, то відняти від *y* число 1.

21. Дано два довільних цілих числа *x* та *y*. Якщо значення суми *x* та *y* більше 10, то у випадку, якщо *x* – непарне число, то помножити його на 4, а якщо ні – поділити його на 2. Якщо значення суми *x* та *y* не більше 10, то помножити *y* на число 9.

22. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* більше, ніж *y*, то у випадку, якщо *x* – непарне число, то додати до нього 1, а якщо ні – поділити його на 2. Якщо значення *x* не більше, ніж *y*, то помножити *y* на число 4.

23. Дано два довільних цілих числа *x* та *y*. Якщо значення *x* менше, ніж *y*, то у випадку, якщо *x* – від'ємне число, то обчислити його модуль, а якщо ні – додати нього 12. Якщо значення *x* не менше, ніж *y*, то помножити *y* на число 3.

24. Дано два довільних цілих числа *x* та *y*. Якщо значення суми *x* та *y* менше 24, то у випадку, якщо *x* – непарне число, то додати до нього число 13, а якщо ні – поділити його на 2. Якщо значення суми *x* та *y* не менше 24, то поділити *y* на число 4.

25. Дано два довільних цілих числа *x* та *y*. Якщо значення x більше, ніж *y*, то у випадку, якщо *x* – парне число, то поділити його на 2, а якщо ні – помножити на 3. Якщо значення *x* не більше, ніж *y*, то додати до *y* число 5.

Завдання 3. Розробити ефективний алгоритм для вирішення завдання (спільний варіант).

Дано три довільних різних цілих числа *x*, *y* та *z*. Вивести всі числа у порядку зростання їх значень, тобто від меншого до більшого. Ефективний алгоритм повинен використовувати мінімальну кількість елементарних логічних операцій, тобто операцій порівняння двох чисел типу $x > y$.

Контрольні запитання до лабораторної роботи 2

1. Який алгоритм називають алгоритмом з розгалуженням?
2. Чи можна замінити структуру вибору за допомогою структур розгалуження?
3. За допомогою якої структури описують процес перетворення даних, що не вимагає перевірки жодних умов чи виконання повторюваних операцій?
4. Чим відрізняється повне розгалуження від неповного?
5. Яке розгалуження називають багатоваріантним вибором?

Лабораторна робота 3
Розроблення схем алгоритмів циклічних обчислювальних процесів

Мета роботи: набути практичних навичок розроблення схем циклічних процесів.

Теоретична частина

Циклічний алгоритм

У деяких алгоритмах передбачено можливість багаторазового виконання деякої сукупності дій. Такі алгоритми називають **циклічними** (циклом), а їх повторювану частину – **тілом циклу**.

Для побудови циклічного алгоритму необхідно:

- визначити всі дії, які необхідно виконати до входу в цикл, тобто провести підготовку циклу;
- визначити всі операції, які ввійдуть до циклу;
- скласти умову виходу з циклу.

Для подання циклічних алгоритмів використовують алгоритмічні конструкції повторення, які реалізуються одним із трьох наведених далі способів.

Прості цикли з параметром. Якщо в процесі перетворення інформації є змінна, значення якої змінюється за відомим правилом, або відомі межі її зміни, то можна визначити кількість повторень ітерацій циклу та організувати вихід із циклічного процесу. Такі цикли називають *циклами з параметром* (арифметичним циклом), а відповідну змінну – *параметром циклу*.

Графічне подання циклу з параметром подане на рис. 3.1.

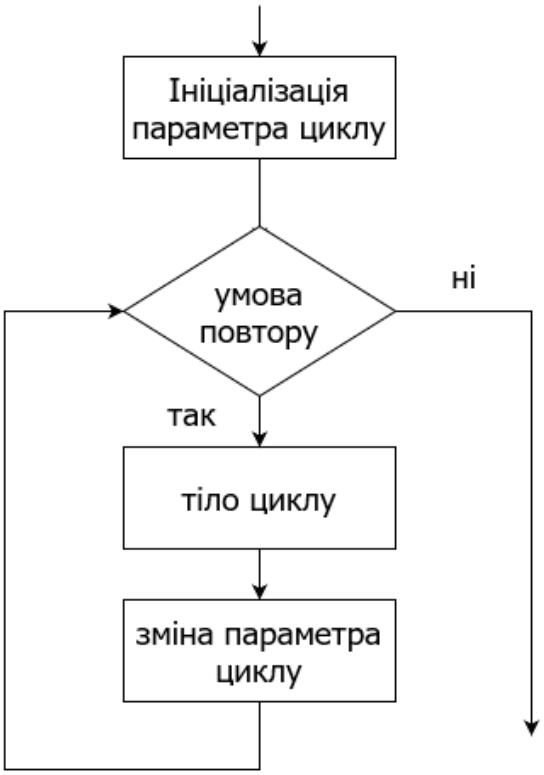


Рис. 3.1. **Блок-схема циклу з параметром**

Як параметр циклу можна використовувати:

- змінну, яка належить до оброблюваної інформації;
- індексну змінну, якщо оброблювана інформація є масивом;
- коефіцієнти, що змінюються за законом арифметичної прогресії.

Наприклад, алгоритм обчислення значення функції:

$$y = \frac{x^2 - 3x + 2}{\sqrt{2x^3 - 1}} \text{ для } x = 1;\ 1.1;\ 1.2;\ \ldots;\ 2$$

представлено на рис. 3.2.

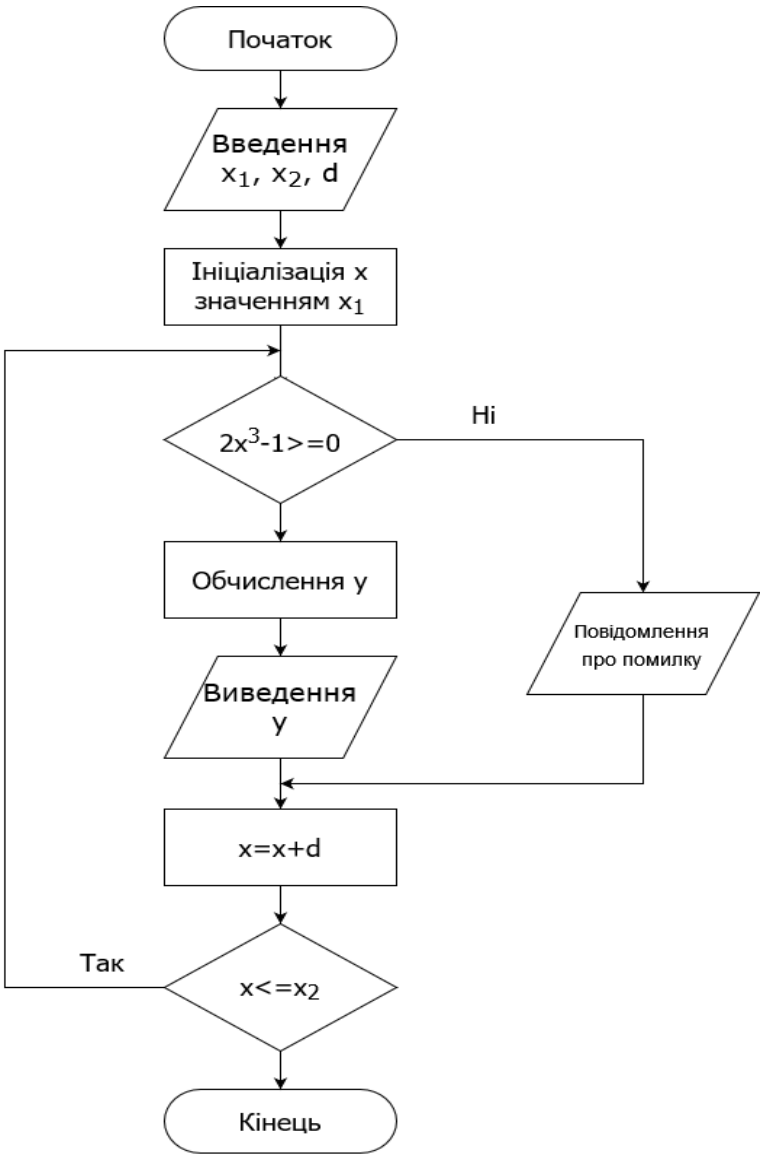


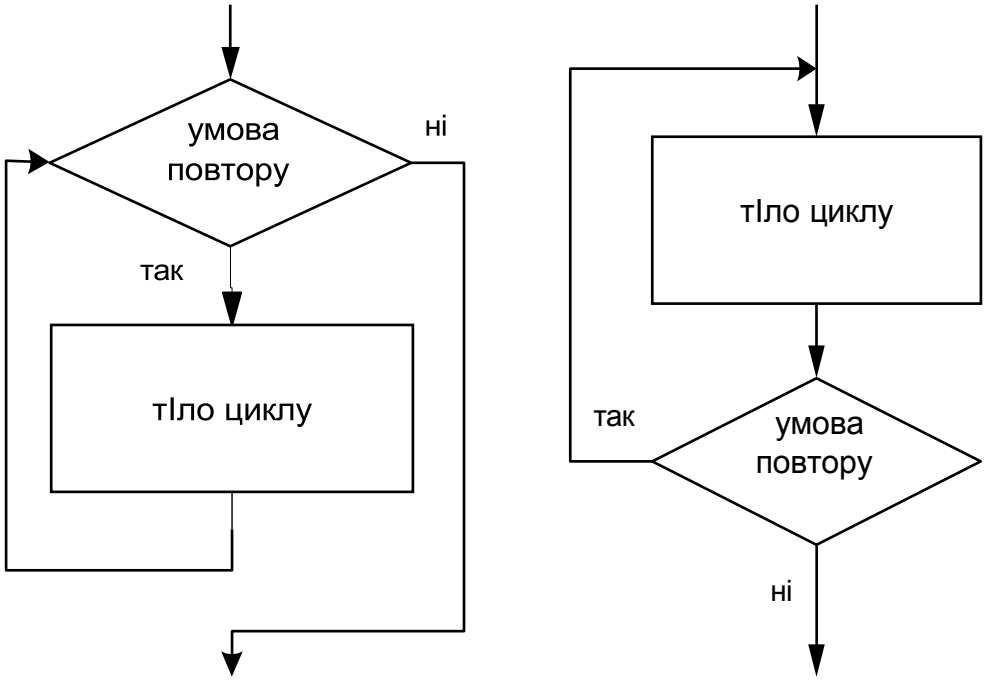
Рис. 3.2. **Приклад простого циклічного процесу**

Тут за параметр циклу можна обрати змінну *x*, яка змінюється від значення x_1 = 1 до значення x_2 = 2 з кроком *d* = 0.1.

Ітераційні цикли. Розв'язання систем лінійних алгебраїчних рівнянь з десятками та сотнями невідомих, пошук коренів алгебраїчних рівнянь високих степенів та коренів трансцендентних рівнянь, розв'язання систем диференційних рівнянь, інтерполяція та екстраполяція функцій, обчислення значень функції за допомогою рядів, інтегрування тощо – усі ці задачі розв'язують за допомогою циклічних алгоритмів, що реалізують циклічні ітераційні процеси, для яких заздалегідь неможливо визначити кількість повторень циклу. Тому необхідно сформулювати умову виходу з циклу з використанням особливостей самої задачі.

Розрізняють два типи ітераційних циклів – з *передумовою* і з *післяумовою*.

У циклі *з передумовою* таку умову перевіряють перед кожною ітерацією циклу (рис. 3.3а), у циклі *з післяумовою* – після ітерації циклу (рис. 3.3б).



а) передумова б) післяумова

Рис. 3.3. **Блок-схеми ітераційних циклів**

Наприклад, треба обчислити з точністю до значення функції $y(x) = e^x$, використовуючи її розкладання у ряд:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^k}{k!} + \frac{x^{k+1}}{(k+1)!} + \ldots = \sum_{k=1}^{\infty} \frac{x^k}{k!}.$$

Відомо, що будь-яку функцію можна наближено подати у вигляді деякого ряду. Підставивши значення аргументу, можна знайти часткову суму ряду і вважати її значенням функції від того самого аргументу з певною точністю наближення. Чим більша точність потрібна, тим більший відрізок ряду треба буде обчислювати.

Якщо поточний елемент ряду позначити через k, то поточне значення суми – через s, доданка суми – через p, точність наближення – через e, коефіцієнт пропорційності обчислювати за формулою:

$$d = \frac{x^{k+1} k!}{(k+1)! x^k} = \frac{x}{k+1},$$

то блок-схема цього алгоритму може бути подана так, як на рис. 3.4.

Практична частина

Система оцінювання. За правильно виконане перше завдання студент отримує 4 бали; за правильно виконані перше та друге завдання – 6 балів; якщо правильно виконано всі три завдання – 7 балів.

Приклад. Вивести за допомогою циклу з передумовою всі числа в діапазоні від 1 до 10, обчислити їх суму.

Оскільки всі дані задані наперед, то їх введення не потрібне. Сума буде накопичуватись у змінній *sum*, яку перед першим виконанням тіла циклу зробимо рівною нулеві. На рис. 3.5 подана блок-схема алгоритму.

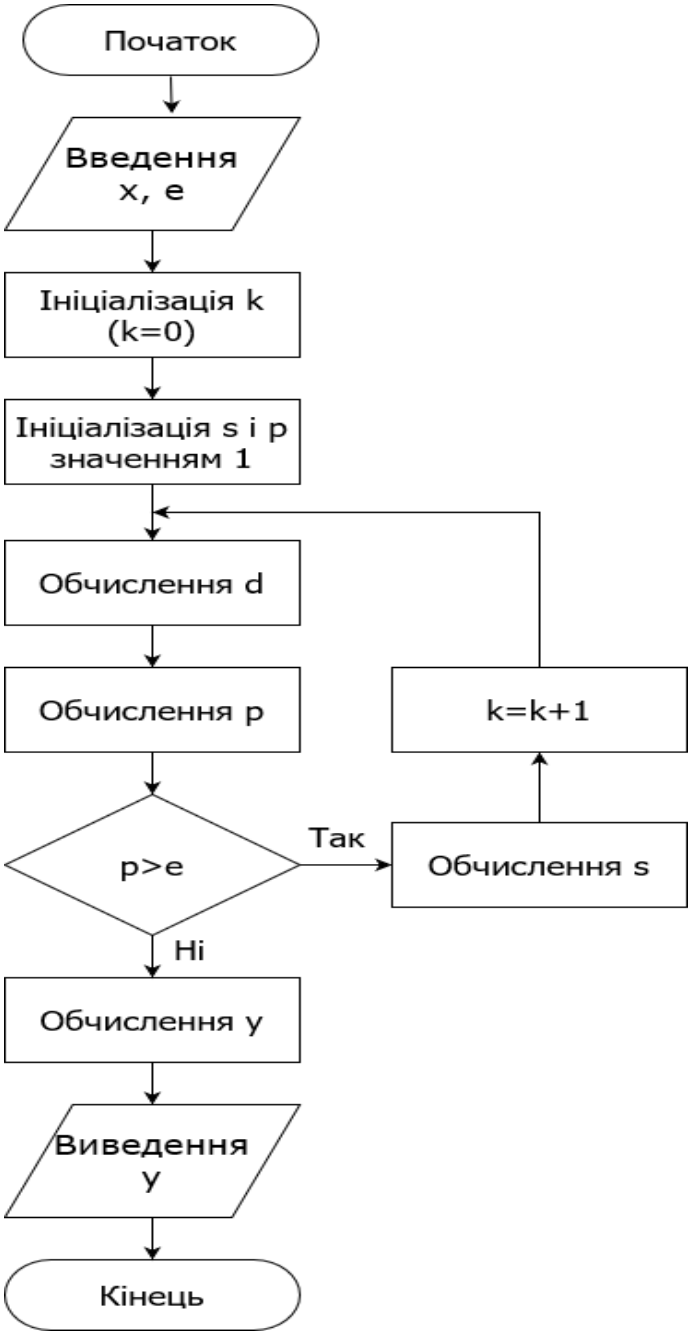


Рис. 3.4. **Блок-схеми вирішення задачі**

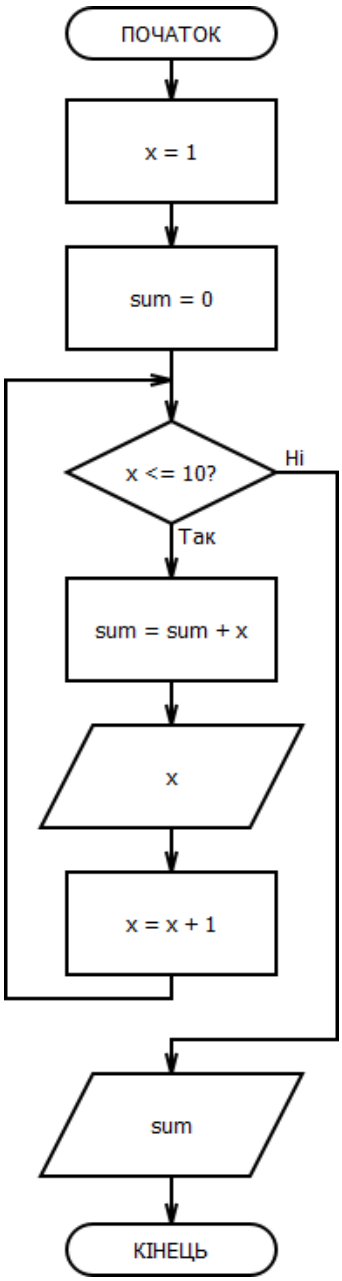


Рис. 3.5. **Блок-схема алгоритму**

Завдання 1. Розробити алгоритм для вирішення завдання (за варіантами):

1. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в діапазоні від 4 до 18 й обчислити їх суму.

2. За допомогою циклу післяумовою вивести в стовпчик усі непарні числа в діапазоні від 3 до 21 й обчислити їх суму.

3. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в зворотному порядку (в порядку спадання) в діапазоні від 16 до 6 й обчислити їх суму.

4. За допомогою циклу післяумовою вивести в стовпчик усі непарні числа в зворотному порядку (в порядку спадання) в діапазоні від 17 до 9 й обчислити їх суму.

5. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в діапазоні від 2 до 16 й обчислити їх суму.

6. За допомогою циклу післяумовою вивести в стовпчик усі непарні числа в діапазоні від 5 до 19 й обчислити їх суму.

7. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в зворотному порядку (в порядку спадання) в діапазоні від 20 до 10 й обчислити їх суму.

8. За допомогою циклу післяумовою вивести в стовпчик усі непарні числа в зворотному порядку (в порядку спадання) в діапазоні від 23 до 11 й обчислити їх суму.

9. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в діапазоні від 4 до 18 й обчислити їх суму.

10. За допомогою циклу післяумовою вивести в стовпчик усі непарні числа в діапазоні від 3 до 21 й обчислити їх суму.

11. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в зворотному порядку (в порядку спадання) в діапазоні від 16 до 6 й обчислити їх суму.

12. За допомогою циклу післяумовою вивести в стовпчик усі непарні числа в зворотному порядку (в порядку спадання) в діапазоні від 17 до 9 й обчислити їх суму.

13. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в діапазоні від 2 до 16 й обчислити їх суму.

14. За допомогою циклу післяумовою вивести в стовпчик усі непарні числа в діапазоні від 5 до 19 й обчислити їх суму.

15. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в зворотному порядку (в порядку спадання) в діапазоні від 20 до 10 й обчислити їх суму.

16. За допомогою циклу післяумовою вивести в стовпчик усі непарні числа в зворотному порядку (в порядку спадання) в діапазоні від 23 до 11 й обчислити їх суму.

17. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в діапазоні від 4 до 18 й обчислити їх суму.

18. За допомогою циклу післяумовою вивести в стовпчик усі непарні числа в діапазоні від 3 до 21 й обчислити їх суму.

19. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в зворотному порядку (в порядку спадання) в діапазоні від 16 до 6 й обчислити їх суму.

20. За допомогою циклу післяумовою вивести в стовпчик усі непарні числа в зворотному порядку (в порядку спадання) в діапазоні від 17 до 9 й обчислити їх суму.

21. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в діапазоні від 2 до 16 й обчислити їх суму.

22. За допомогою циклу післяумовою вивести в стовпчик усі непарні числа в діапазоні від 5 до 19 й обчислити їх суму.

23. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в зворотному порядку (в порядку спадання) в діапазоні від 20 до 10 й обчислити їх суму.

24. За допомогою циклу післяумовою вивести в стовпчик усі непарні числа в зворотному порядку (в порядку спадання) в діапазоні від 23 до 11 й обчислити їх суму.

25. За допомогою циклу з передумовою вивести в стовпчик усі парні числа в діапазоні від 4 до 18 й обчислити їх суму.

Завдання 2. Розробити алгоритм для вирішення завдання (за варіантами):

1. Визначити добуток ненульових елементів одновимірного масиву між першим додатнім та останнім непарним елементами масиву включно.

2. Визначити суму парних елементів одновимірного масиву між першим та останнім непарними елементами масиву включно.

3. Визначити кількість непарних елементів одновимірного масиву між першим від'ємним та останнім парним елементами масиву включно.

4. Визначити добуток ненульових елементів одновимірного масиву між мінімальним та максимальним елементами масиву включно.

5. Визначити добуток ненульових елементів одновимірного масиву між першим непарним та останнім від'ємним елементами масиву включно.

6. Визначити суму непарних елементів одновимірного масиву між першим парним та останнім додатнім елементами масиву включно.

7. Визначити кількість парних елементів одновимірного масиву між мінімальним та останнім непарним елементами масиву включно.

8. Визначити добуток ненульових елементів одновимірного масиву між першим додатнім та максимальним елементами масиву включно.

9. Визначити суму додатних елементів одновимірного масиву між першим та останнім непарними елементами масиву включно.

10. Визначити кількість ненульових елементів одновимірного масиву між першим та другим від'ємними елементами масиву включно.

11. Визначити добуток ненульових елементів одновимірного масиву між першим додатнім та останнім непарним елементами масиву включно.

12. Визначити суму парних елементів одновимірного масиву між першим та останнім непарними елементами масиву включно.

13. Визначити кількість непарних елементів одновимірного масиву між першим від'ємним та останнім парним елементами масиву включно.

14. Визначити добуток ненульових елементів одновимірного масиву між мінімальним та максимальним елементами масиву включно.

15. Визначити добуток ненульових елементів одновимірного масиву між першим непарним та останнім від'ємним елементами масиву включно.

16. Визначити суму непарних елементів одновимірного масиву між першим парним та останнім додатнім елементами масиву включно.

17. Визначити кількість парних елементів одновимірного масиву між мінімальним та останнім непарним елементами масиву включно.

18. Визначити добуток ненульових елементів одновимірного масиву між першим додатнім та максимальним елементами масиву включно.

19. Визначити суму додатних елементів одновимірного масиву між першим та останнім непарними елементами масиву включно.

20. Визначити кількість ненульових елементів одновимірного масиву між першим та другим від'ємними елементами масиву включно.

21. Визначити добуток ненульових елементів одновимірного масиву між першим додатнім та останнім непарним елементами масиву включно.

22. Визначити суму парних елементів одновимірного масиву між першим та останнім непарними елементами масиву включно.

23. Визначити кількість непарних елементів одновимірного масиву між першим від'ємним та останнім парним елементами масиву включно.

24. Визначити добуток ненульових елементів одновимірного масиву між мінімальним та максимальним елементами масиву включно.

25. Визначити добуток ненульових елементів одновимірного масиву між першим непарним та останнім від'ємним елементами масиву включно.

Завдання 3. Розробити алгоритм для вирішення завдання (спільний варіант).

Задано довільне ціле число. Визначити кількість нулів у запису цього числа та зайти найбільшу цифру.

Контрольні запитання до лабораторної роботи 3

1. Під час виконання якого із циклів тіло циклу буде виконуватися принаймні один раз?
2. За умови використання якого типу циклу можлива ситуація, коли команди, що входять до тіла циклу, не будуть виконані жодного разу?
3. Чи може цикл з параметром бути замінений за допомогою циклу з передумовою?
4. Що називають тілом циклу?
5. Що називають параметром циклу?

Лабораторна робота 4
Розроблення програм для машини Поста

Мета роботи:

- вивчити машину Поста і способи написання алгоритмів для неї;
- отримати навички з написання алгоритмів виконання різних завдань для машини Поста.

Теоретична частина

1. Будова машини Поста

Машина Поста – це абстрактна машина, яка складається зі стрічки і каретки (головки, яка зчитує і записує). Стрічка нескінченна і розділена на секції однакового розміру. У кожну секцію стрічки заносять один символ двійкової інформації, який підлягає обробленню. Один із символів двійкового алфавіту – мітка "●" (або "1"), інший – порожнеча (або "0"). Якщо в секцію занесена "●" – секція відмічена, якщо в секції "●" немає – секція порожня або невідмічена.

Каретка може пересуватися уздовж стрічки ліворуч або праворуч. Коли вона нерухома і стоїть навпроти рівно однієї секції стрічки, то кажуть, що каретка оглядає цю секцію. А таку секцію називають поточною або таку, яку оглядають.

За одиницю часу, яку називається кроком, каретка може зрушити на одну секцію ліворуч або праворуч. Крім того, каретка може також

розпізнати, чи є мітка в секції, яку вона оглядає, вона може заносити мітку в порожню секцію і може видаляти мітку із відміченої секції.

Команди, за якими каретка повинна занести мітку до відміченої секції або видалити мітку з порожньої секції є неприпустимими.

2. Система команд машини Поста

Формат команди машини Поста має вигляд:

$$n\ K\ m,$$

де: n – номер поточної команди;

K – команда з системи команд машини Поста;

m – посилання – номер команди, яка буде виконана наступною.

Послідовність команд з системи команд становить програму, якщо:

1) на *n*-му місці цієї програми буде стояти команда з номером *n*;
2) посиланню *m* відповідає реальна команда в програмі.

3. Тренажер "Машина Поста"

Тренажер "Машина Поста" – це навчальна модель універсального виконавця (абстрактної обчислювальної машини), заснована на роботах Поста з уточнення поняття алгоритму (рис. 4.1). Згідно з тезою Поста, будь-який алгоритм може бути записаний у вигляді програми для машини Поста.

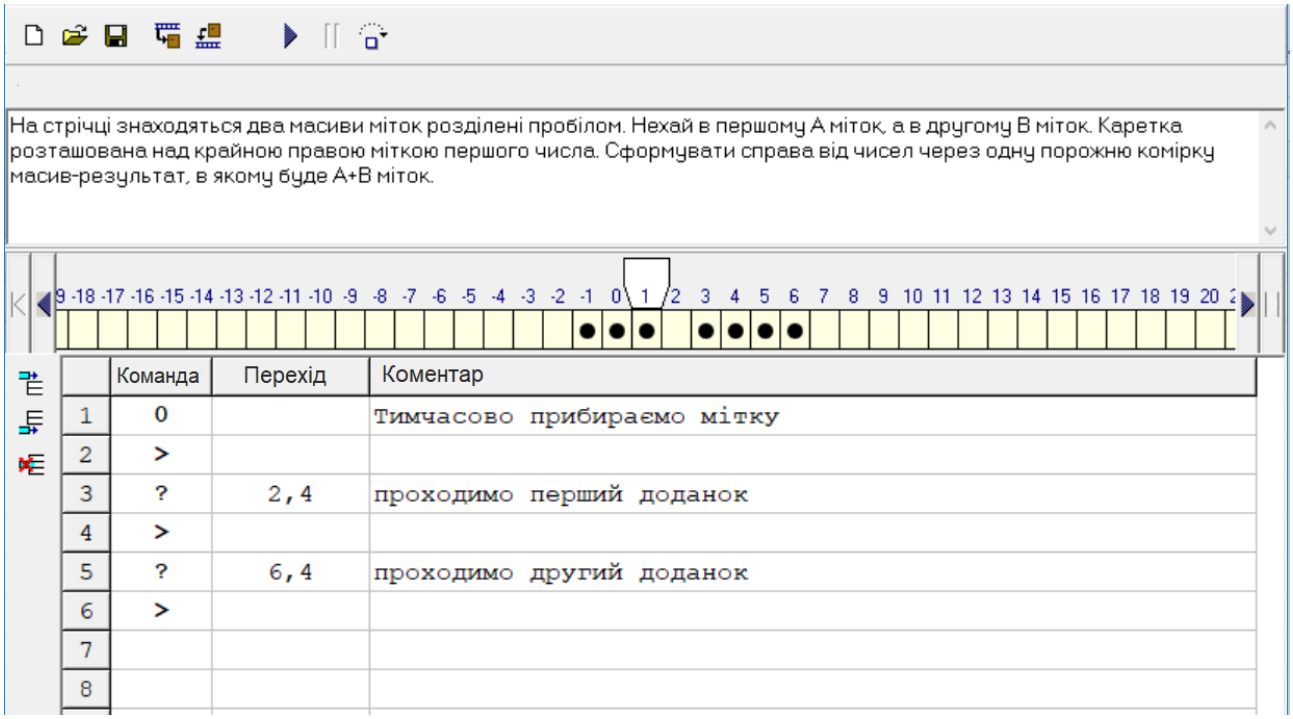


Рис. 4.1. **Тренажер "Машина Поста"**

Машина Поста складається з каретки (зчитувальної і записувальної головки) і нескінченної стрічки, розподіленою на секції (комірки). Кожна секція стрічки може бути або порожньою ("0"), або містити мітку ("1").

Програма складається з пронумерованих рядків. У кожному рядку записують одну з таких команд (табл. 4.1).

Номер рядка переходу в командах **>**, **<**, **0** і **1** можна не вказувати, при цьому відбувається перехід до наступного рядка.

Для завершення роботи програми досить зробити перехід на рядок 0, наприклад, так:

? 25, 0 – зупинити програму, якщо поточна комірка містить "1", інакше перейти до рядка 25.

Таблиця 4.1

Команди машини Поста

Команда	Зміст команди
> N	Перемістити каретку праворуч на 1 клітинку і перейти до рядка з номером N
< N	Перемістити каретку ліворуч на 1 клітинку і перейти до рядка з номером N
0 N	Записати в поточну секцію "0" (стерти мітку) і перейти до рядка з номером N
1 N	Записати в поточну секцію "1" (поставити мітку) і перейти до рядка з номером N
? N, M	Якщо поточна секція містить "0" (не позначена), то перейти до рядка з номером N, інакше перейти до рядка M
.	Зупинити програму

4. Порядок роботи з тренажером "Машина Поста"

У верхній частині програми знаходиться поле редактора, в яке можна ввести умову задачі у вільній формі.

Стрічка переміщається ліворуч і праворуч за допомогою кнопок, розташованих ліворуч і праворуч від неї. Подвійним клацанням по секції стрічки можна змінити її вміст (у "класичному" варіанті замінюється "0" на "1" або навпаки, в трійковій машині Поста символи "пропуск", "0" і "1" під час послідовних подвійних клацаннях змінюються циклічно).

За допомогою меню "Стрічка" можна запам'ятати стан стрічки у внутрішньому буфері і відновити стрічку з буфера.

За допомогою меню "Алфавіт" можна перемикати режим роботи машини (двійковий або трійковий алфавіт). У класичному режимі меню "Вид" дозволяє налаштувати вигляд стрічки (точки, позначки-галочки, двійковий код).

У таблиці в нижній частині вікна набирають програму. У першому стовпці записані номери рядків, він заповнюється автоматично. У другому стовпці зі списку вибирають потрібну команду, а в третьому вводять номер рядка для переходу (якщо це необхідно). Для додавання команди в другому стовпці можна використовувати "спливаюче" контекстне меню.

Четвертий стовпець може містити коментар до кожного рядка програми. Додати і видалити рядки таблиці можна за допомогою кнопок, розташованих ліворуч від таблиці.

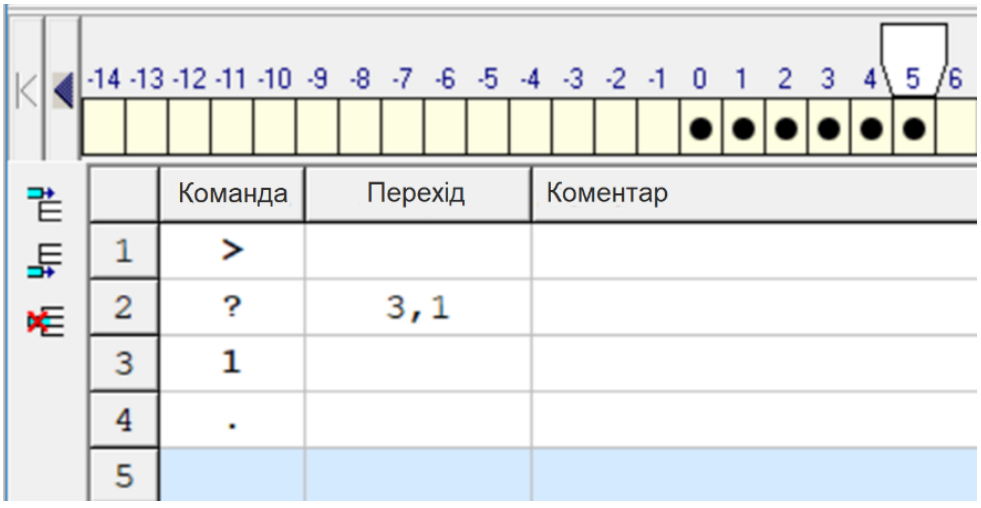
Програма може виконуватися безперервно (F9) або по етапах (F8). Команда, яка зараз буде виконуватися, підсвічується зеленим фоном. Швидкість виконання регулюють за допомогою меню "Швидкість".

Завдання для машини Поста можна зберігати в файлах. Зберігається умова завдання, програма, стан стрічки і вид позначок. Під час завантаження завдання з файла і збереження в файлі початковий стан стрічки автоматично записується в буфер.

5. Приклади розв'язання задач

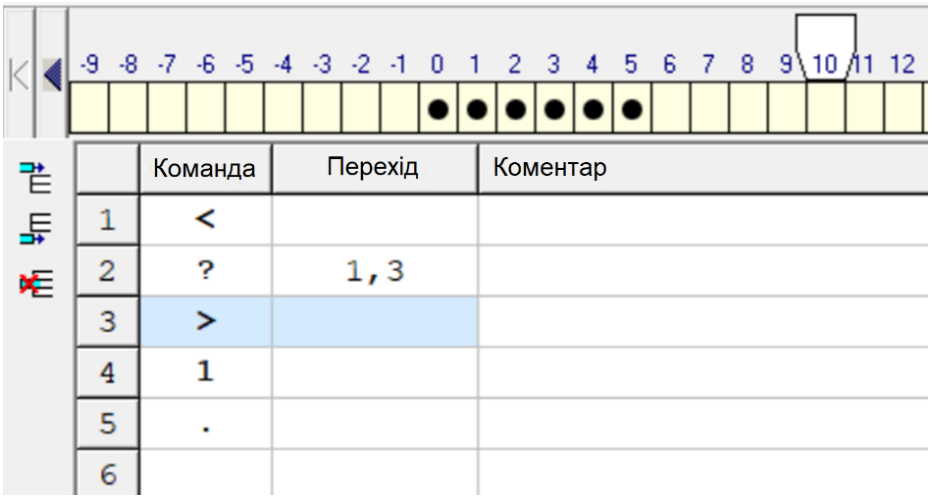
Приклад 1. Число записано в унарній системі, каретка стоїть над довільною позначкою у запису числа. Збільшити число на 1.

Рішення:



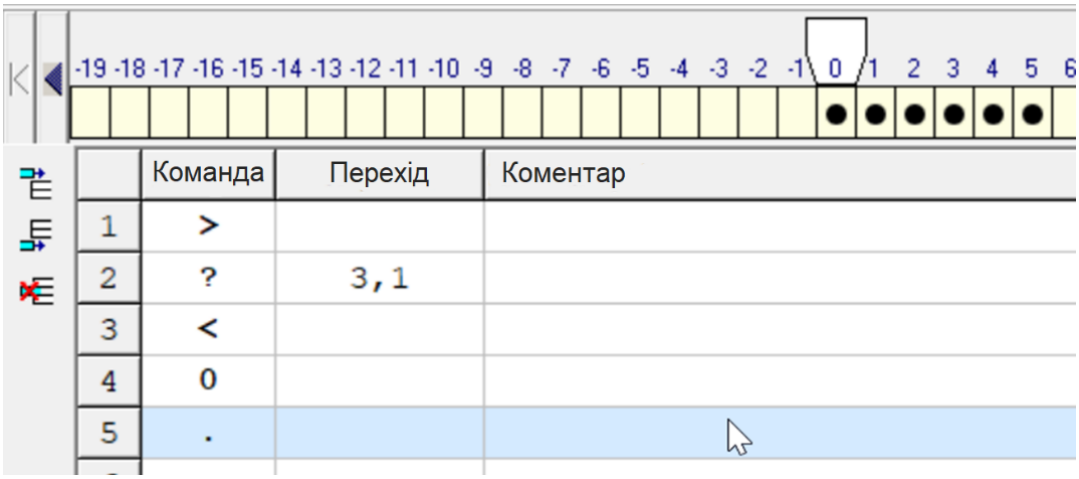
Приклад 2. Число записано в унарній системі, каретка стоїть десь праворуч від запису числа. Збільшити число на 1.

Рішення:



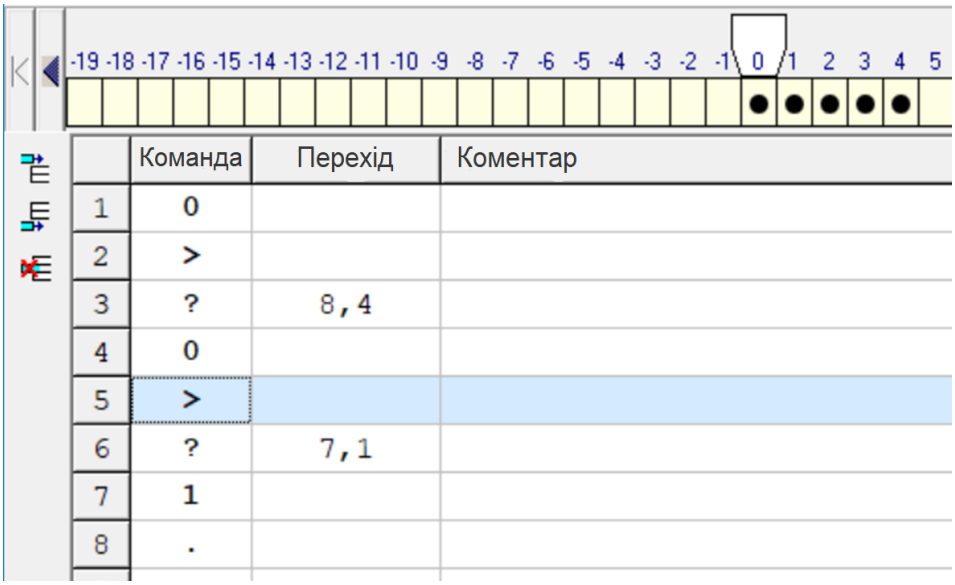
Приклад 3. Число записано в унарній системі, каретка стоїть над першою відміткою. Зменшити число на 1.

Рішення:



Приклад 4. Ціле число N записано на стрічці в унарній системі числення (як послідовність з N міток). Обчислити функцію, яка дорівнює 1, якщо число N парне, і дорівнює 0 (залишає порожню стрічку), якщо N – непарне. У початковий момент каретка стоїть над першою зліва міткою.

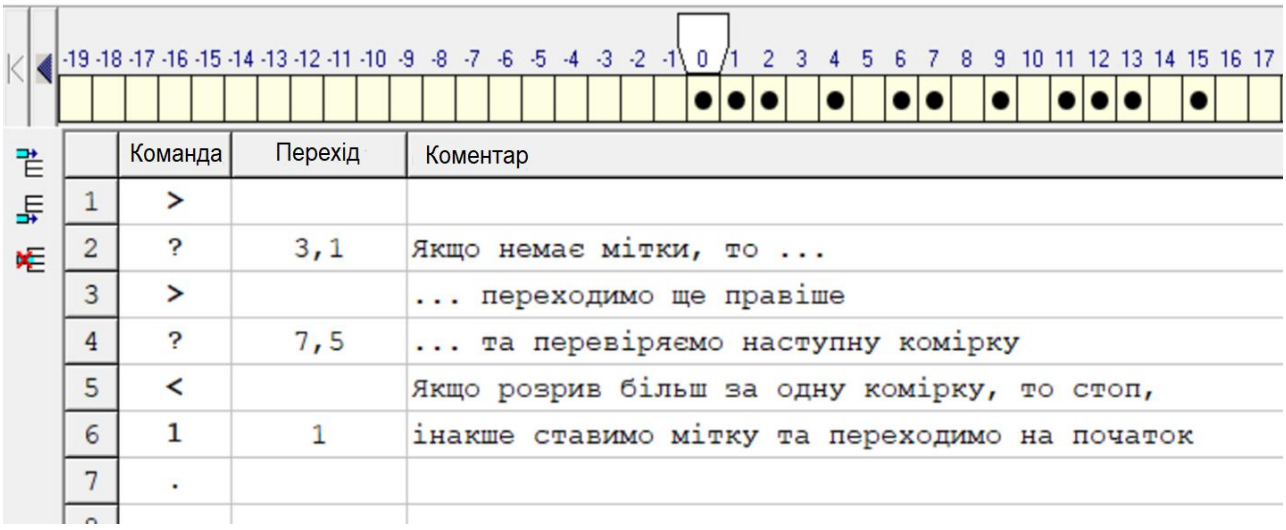
Рішення:



	Команда	Перехід	Коментар
1	0		
2	>		
3	?	8,4	
4	0		
5	>		
6	?	7,1	
7	1		
8	.		

Приклад 5. На стрічці розставлені мітки, між якими можуть бути пропуски довжиною в одну клітинку. Заповнити всі пропуски мітками. Каретка стоїть над першою зліва міткою.

Рішення:



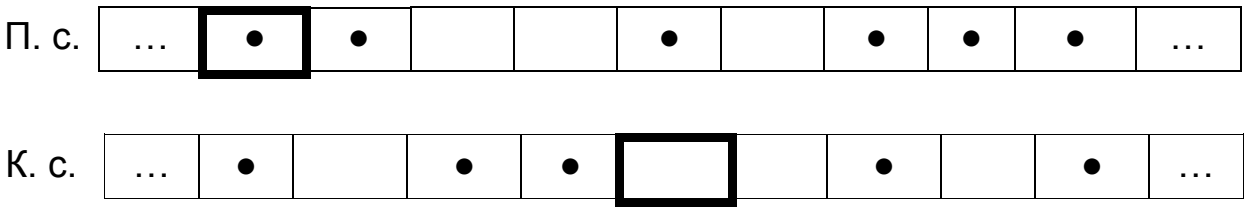
	Команда	Перехід	Коментар
1	>		
2	?	3,1	Якщо немає мітки, то ...
3	>		... переходимо ще правіше
4	?	7,5	... та перевіряємо наступну комірку
5	<		Якщо розрив більш за одну комірку, то стоп,
6	1	1	інакше ставимо мітку та переходимо на початок
7	.		

Практична частина

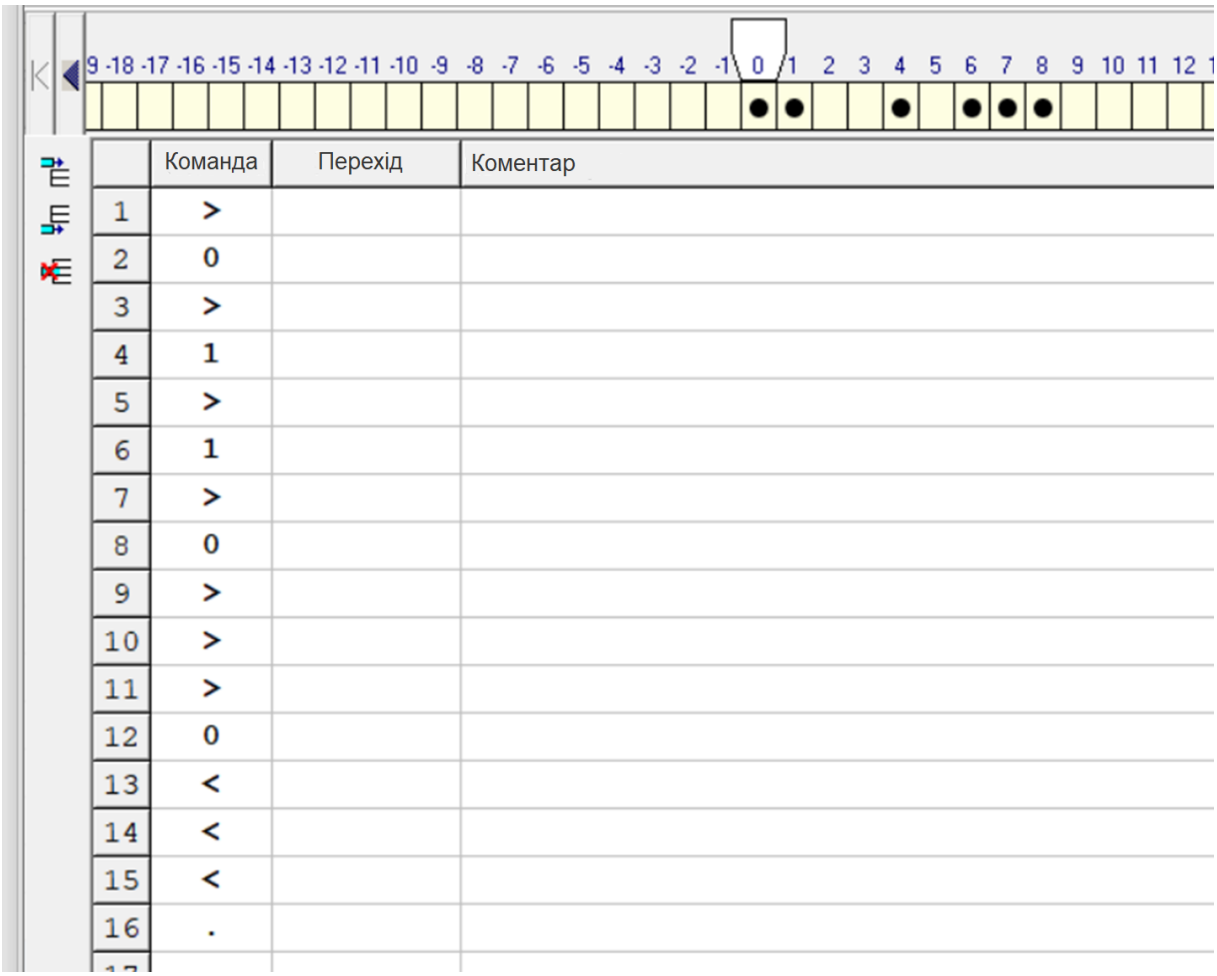
Розроблення програм для машини Поста

Система оцінювання. За правильно виконане перше завдання студент отримує 4 бали; за правильно виконані перше та друге завдання – 6 балів; якщо правильно виконано усі три завдання – 7 балів.

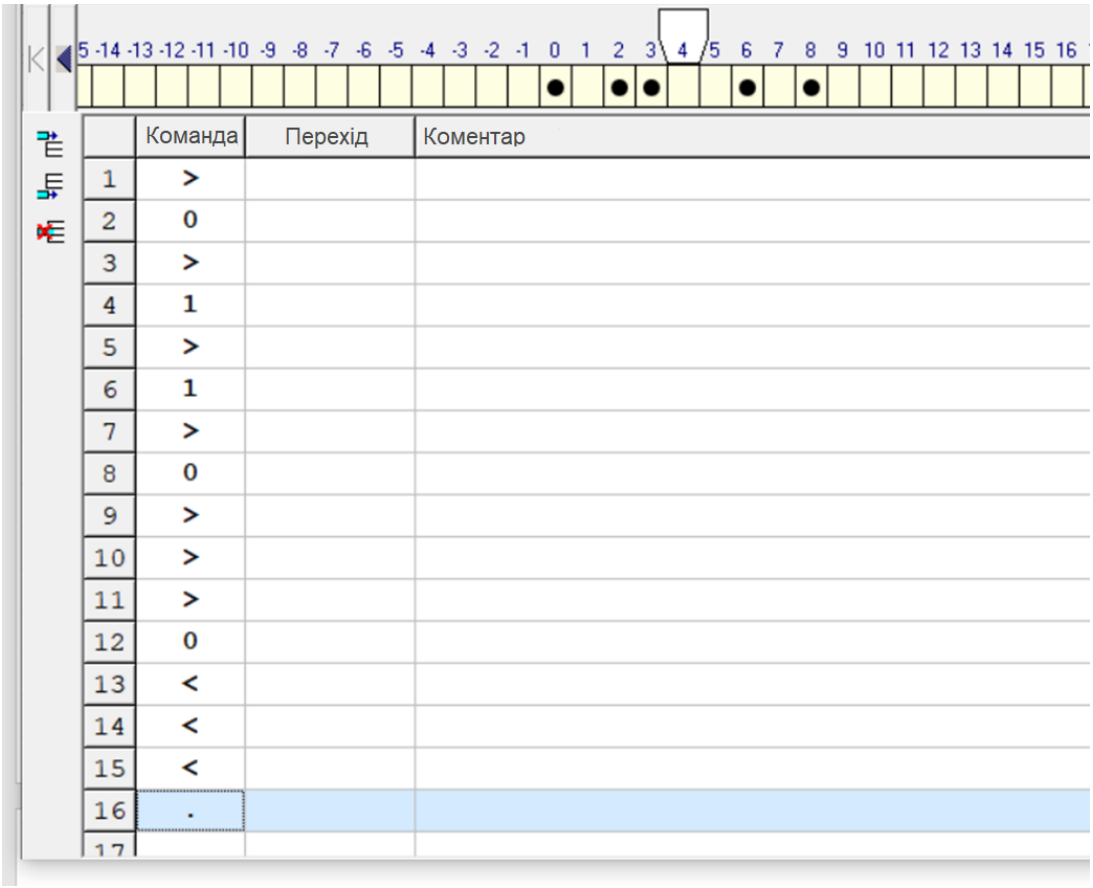
Приклад. Скласти програму для машини Поста для переведення інформаційної стрічки з початкового стану (п. с.) у кінцевий стан (к. с.), де чорним прямокутником вказано положення каретки.



Початковий стан та програма для машини Поста:

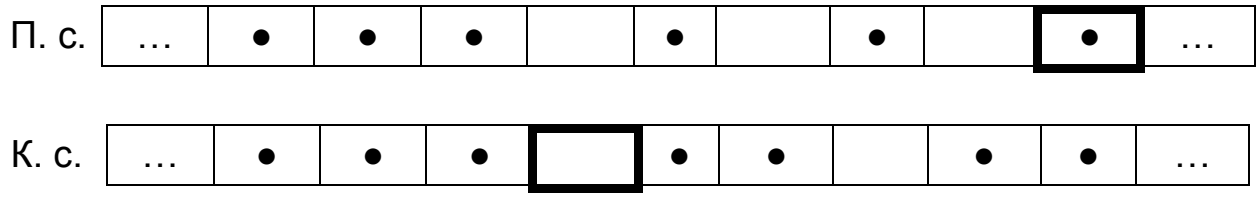


Кінцевий стан після виконання програми:

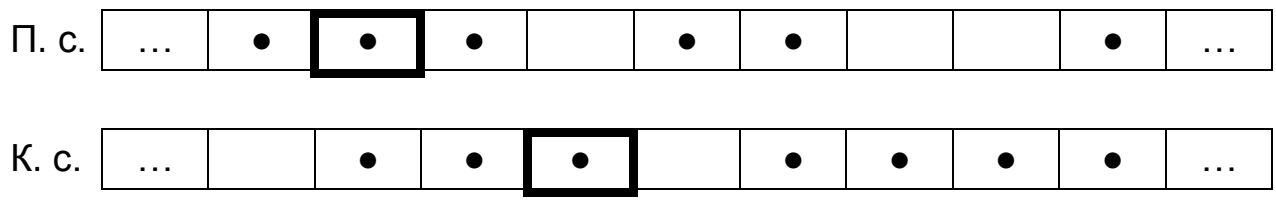


Завдання 1. Скласти програму для машини Поста для переведення інформаційної стрічки з початкового стану (п. с.) у кінцевий стан (к. с.), де чорним прямокутником вказано положення каретки (за варіантами):

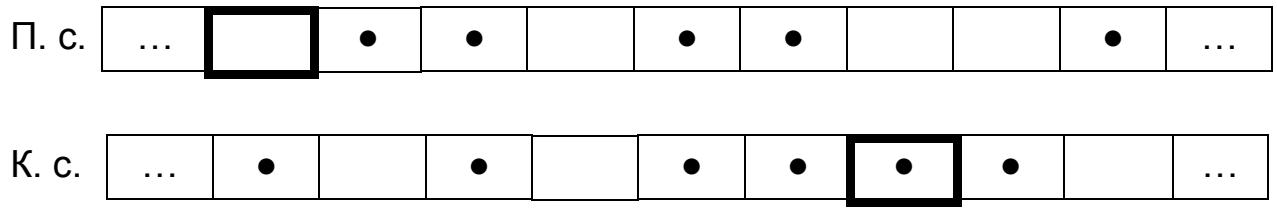
1-й варіант



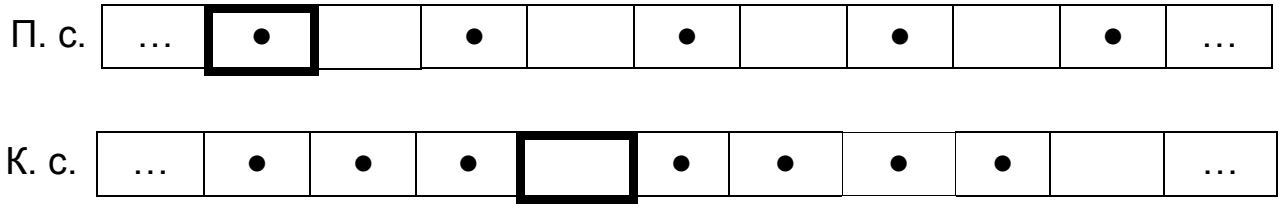
2-й варіант



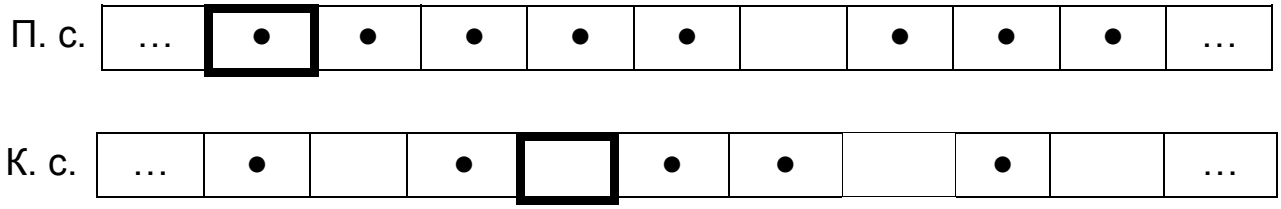
3-й варіант



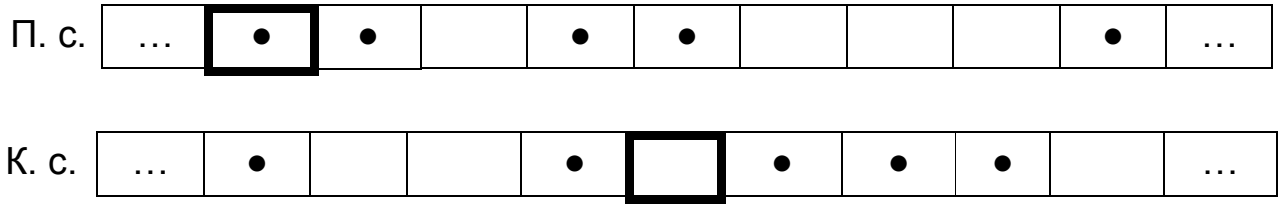
4-й варіант



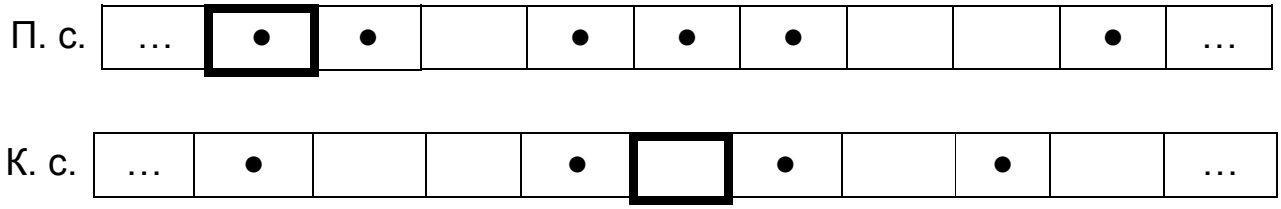
5-й варіант



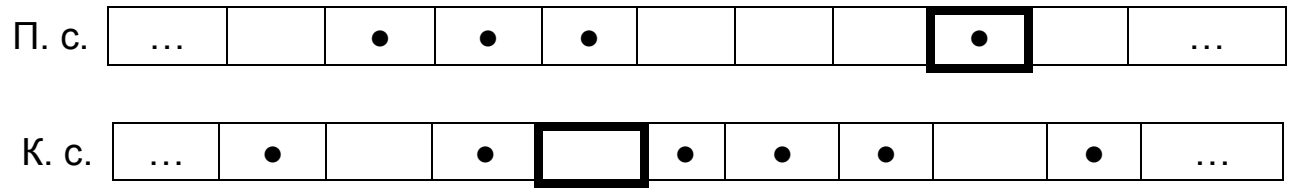
6-й варіант



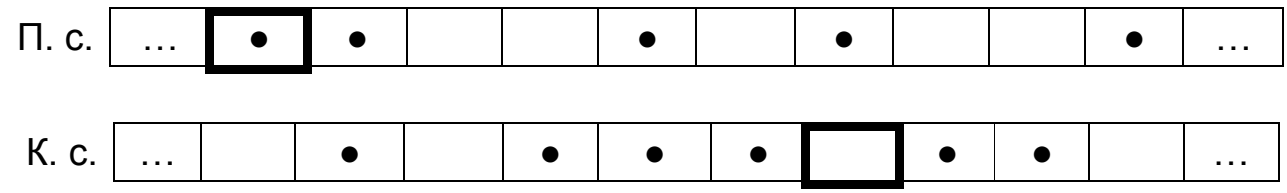
7-й варіант



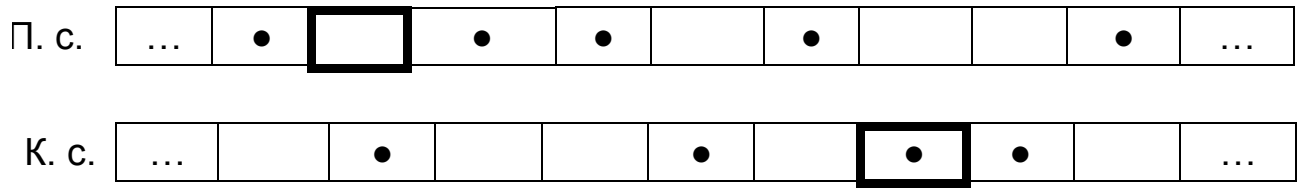
8-й варіант



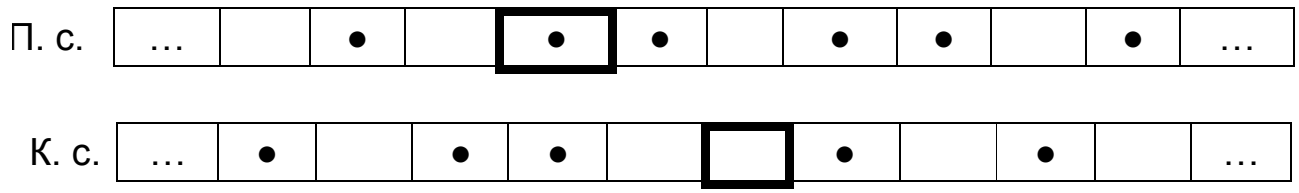
9-й варіант



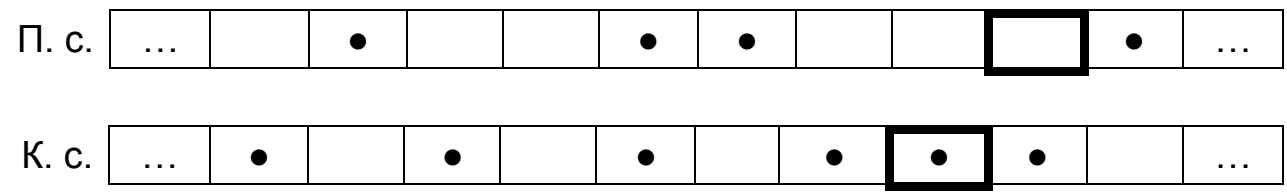
10-й варіант



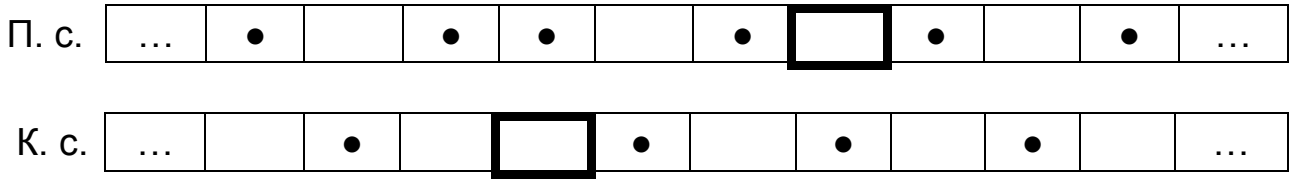
11-й варіант



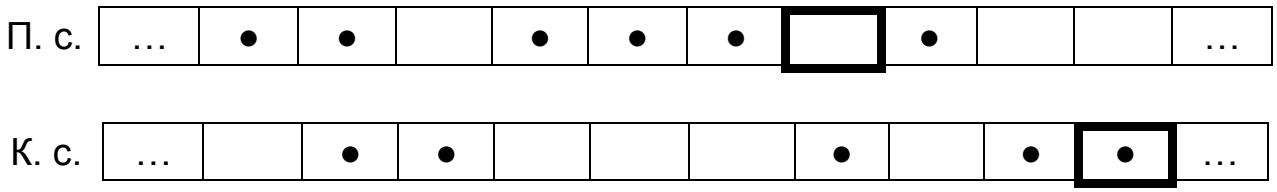
12-й варіант



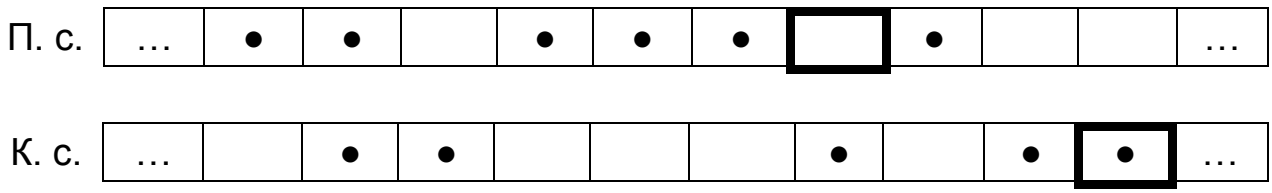
13-й варіант



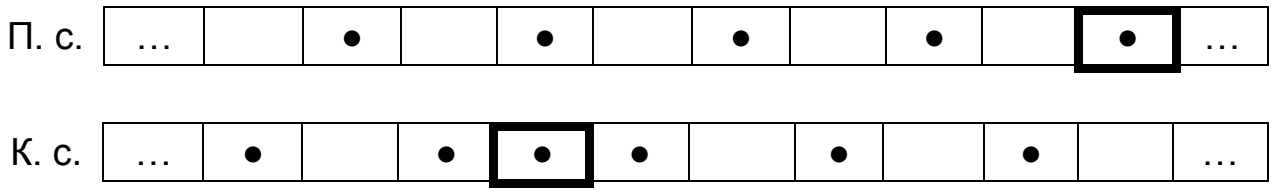
14-й варіант



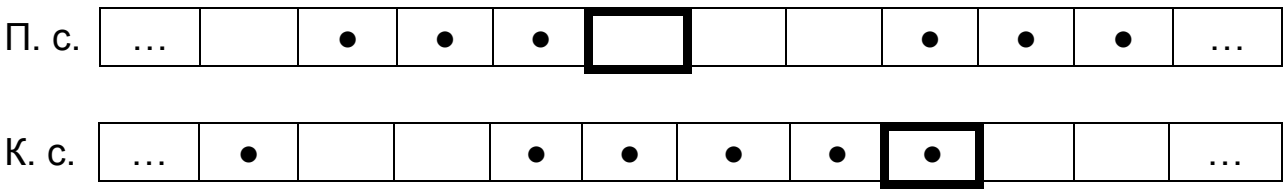
15-й варіант



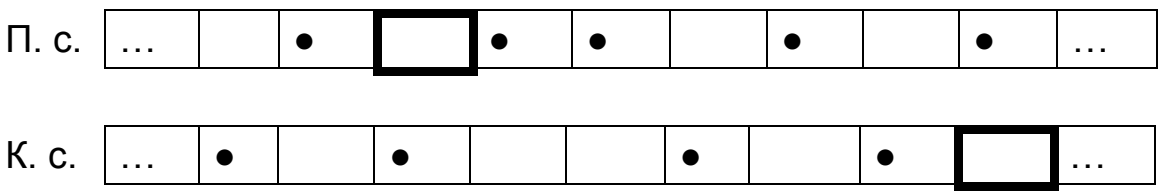
16-й варіант



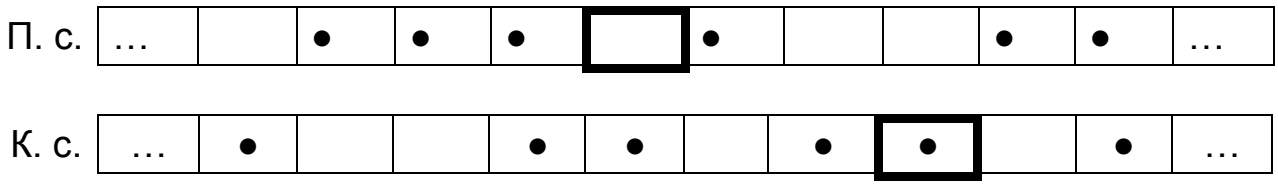
17-й варіант



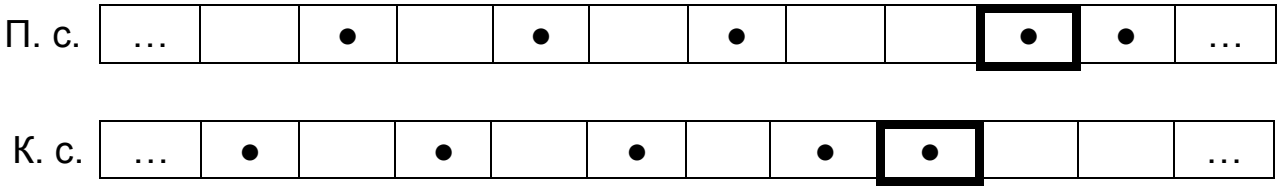
18-й варіант



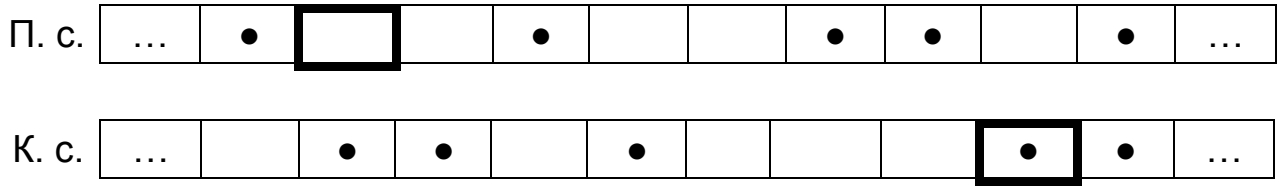
19-й варіант



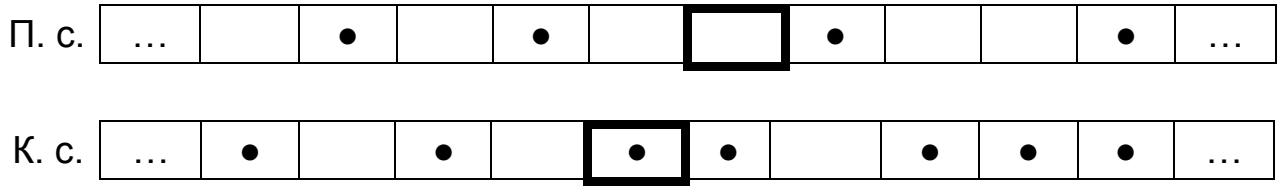
20-й варіант



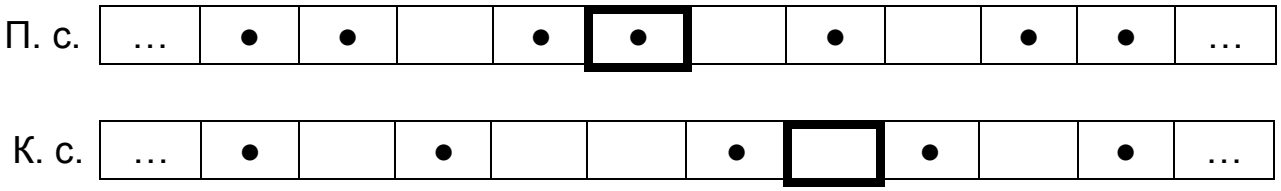
21-й варіант



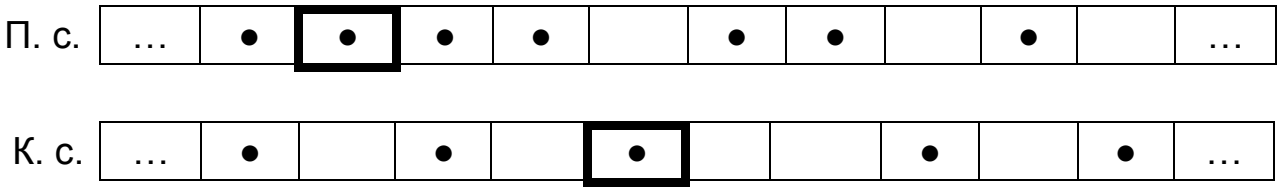
22-й варіант



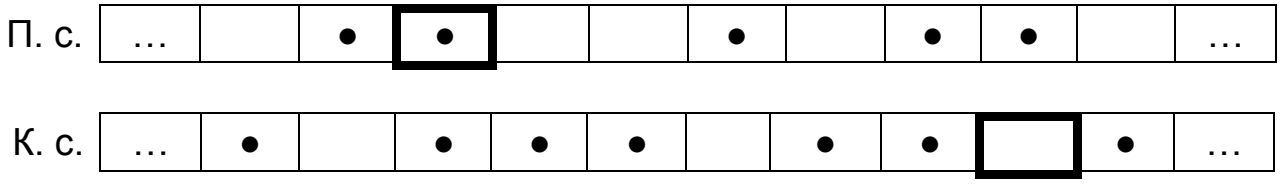
23-й варіант



24-й варіант



25-й варіант



Завдання 2. Скласти програму для машини Поста (за варіантами).

1. На стрічці машини Поста розташовано масив з *N* міток. Скласти програму, діючи за якою, машина з'ясує, чи ділиться число на 3. Якщо так, то після масиву через одну пусту секцію поставити мітку.

2. На стрічці машини Поста розташовано масив з *N* міток. Скласти програму, діючи за якою, машина з'ясує, чи ділиться число на 2. Якщо так, то після масиву через одну пусту секцію поставити мітку.

3. На стрічці машини Поста задано довільну послідовність міток. Скласти програму, яка видалить усі мітки цієї послідовності, окрім крайніх. Каретки розташувати над крайньою справа міткою.

4. На стрічці машини Поста розташовано масив з *N* міток. Скласти програму, діючи за якою, машина з'ясує, чи є число *N* непарним. Якщо так, то після масиву через одну пусту секцію поставити мітку.

5. Напишіть програму, що подвоює довільне число, що розташовано на стрічці машини Поста, додаючи мітки справа від нього. Каретка знаходиться над крайньою лівою міткою числа. Первісне число лишається на місці.

6. На стрічці машини Поста розташовано масив з *N* міток (мітки розташовані через пробіл). Потрібно стиснути масив так, щоб всі *n* міток займали *N* розташованих поруч комірок.

7. Скласти програму для машини Поста, яка буде визначати залишок від цілочислового ділення числа на стрічці на число 3. Результат записати справа від числа через одну комірку.

8. На стрічці машини Поста розташовано масив з *N* міток (мітки розташовано поспіль). Потрібно перетворити масив так, щоб усі *n* міток були розміщені через один пробіл.

9. На стрічці машини Поста розташовано масив з 2 × *N* заповнених комірок. Скласти програму, за якою машина Поста розділить на відстань в 1 комірку дві половини цього масиву.

10. Скласти програму додавання двох цілих додатних чисел, записаних на стрічці машини Поста на відстані деякої кількості пустих комірок одна від одної. Каретка знаходиться над крайньою лівою міткою лівого числа.

11. На стрічці машини Поста розташовано масив з 2 × *N* заповнених комірок. Скласти програму, за якою машина Поста розділить на відстань в 3 комірки дві половини цього масиву.

12. На стрічці машини Поста записані два цілих числа *N* (зліва) та *M* (справа), що розділені 1 пробілом. Знайти суму цих чисел, додавши

у кінець правого числа кількість міток лівого числа. Каретка знаходиться над пробілом, що розділяє числа.

13. На стрічці машини Поста записані два цілих числа *N* (зліва) та *M* (справа), що розділені 1 пробілом. Знайти суму цих чисел, додавши у початок лівого числа кількість міток правого числа. Каретка знаходиться над пробілом, що розділяє числа.

14. На стрічці машини Поста записані два цілих числа *N* (зліва) та *M* (справа), що розділені 1 пробілом (*N* > *M*). Знайти різницю *N* – *M* цих чисел. Каретка знаходиться над крайньою лівою міткою лівого числа.

15. На стрічці машини Поста записані два цілих числа *N* (зліва) та *M* (справа), що розділені 1 пробілом (*N* < *M*). Знайти різницю *M* – *N* цих чисел. Каретка знаходиться над крайньою лівою міткою лівого числа.

16. Напишіть програму, що подвоює довільне число, що розташоване на стрічці машини Поста, додаючи мітки зліва від нього. Каретка знаходиться над крайньою правою міткою числа. Первісне число лишається на місці.

17. На стрічці машини Поста записані два цілих числа *N* (зліва) та *M* (справа), що розділені 1 пробілом. З'ясувати, яке з двох чисел більше? Якщо праве, то залишити одну мітку на стрічці, а якщо ліве, то стерти всі мітки. Каретка знаходиться над пробілом, що розділює числа.

18. Скласти програму віднімання двох цілих додатних чисел, записаних на стрічці машини Поста на відстані деякої кількості пустих комірок одна від одної. Каретка знаходиться над крайньою лівою міткою лівого числа.

19. На стрічці машини Поста записані два цілих числа *N* (зліва) та *M* (справа), що розділені 1 пробілом (*N* > 2 × *M*). Обчислити значення виразу *N* – 2 × *M*. Каретка знаходиться над крайньою лівою міткою лівого числа.

20. На стрічці машини Поста записані два цілих числа *N* (зліва) та *M* (справа), що розділені 1 пробілом (2 × *N* < *M*). Обчислити значення виразу *M* – 2 × *N*. Каретка знаходиться над крайньою лівою міткою лівого числа.

21. Заданий масив міток. Каретка розташовується десь над масивом, але не над крайніми мітками. Стерти всі мітки, окрім крайніх, та повернути каретку в первісне положення.

22. Подвоїти даний масив міток справа від нього, через одну комірку і потім стерти первісний масив. Каретка знаходиться над крайньою зліва міткою.

23. Подвоїти даний масив міток зліва від нього, через одну комірку і потім стерти первісний масив. Каретка знаходиться над крайньою справа міткою.

24. На стрічці машини Поста записані два цілих числа *N* (зліва) та *M* (справа), що розділені 1 пробілом. З'ясувати, яке з двох чисел більше? Якщо ліве, то залишити одну мітку на стрічці, а якщо праве, то стерти всі мітки. Каретка знаходиться над пробілом, що розділює числа.

25. На стрічці машини Поста задана послідовність з 2 × *N* + 1 міток. Знайти середню мітку послідовності і стерти її.

Завдання 3. Скласти програму для машини Поста.

На стрічці машини Поста знаходиться n масивів міток, після останнього масиву на відстані більш ніж трьох порожніх комірок знаходиться 1 мітка. Масиви розділені трьома порожніми комірками. Кількість міток у масивах не може бути менше двох. Провести таке оброблення масивів: якщо кількість міток у масиві є кратною трьом, то стерти мітки в цьому масиві через одну, інакше масив стерти повністю. Каретка знаходиться над крайньою зліва міткою першого масиву.

Контрольні запитання до лабораторної роботи 4

1. Скільки команд має машина Поста?
2. В якій системі числення можна подати цілі числа в машині Поста?
3. З яких елементів складається машина Поста?
4. Чи припустимою є команда видалення мітки, якщо поточна комірка є порожньою?
5. З чого складається програма, яка буде виконуватися на машині Поста?

Лабораторна робота 5
Розроблення програм для машини Тюринга

Мета роботи:

- вивчити машину Тюринга і способи написання алгоритмів для неї;
- отримати навички з написання алгоритмів вирішення різних завдань для машини Тюринга.

Теоретична частина

1. Будова машини Тюринга

Машина Тюринга складається з каретки (головки, що зчитує і записує) і нескінченної стрічки, розбитої на комірки. Кожна комірка стрічки може містити символ з деякого алфавіту $\mathbf{A = \{a_0, a_1, \ldots, a_N\}}$. Будь-алфавіт містить символ "пропуск", який позначається як $\mathbf{a_0}$ або Λ. Під час введення команд пробіл замінюється знаком підкреслення "_".

Машина Тюринга – це автомат, яким управляє таблиця. Рядки в таблиці відповідають символам обраного алфавіту А, а стовпчики – станам автомата $\mathbf{Q = \{q_0, q_1, \ldots, q_M\}}$. На початку роботи машина Тюринга знаходиться в стані $\mathbf{q_1}$. Стан $\mathbf{q_0}$ – це кінцевий стан: потрапивши в нього, автомат закінчує роботу.

У кожній клітині таблиці, що відповідає деякому символу $\mathbf{a_i}$ і деякому стану $\mathbf{q_j}$, знаходиться команда, що складається з трьох частин:

1. Символ з алфавіту А;
2. Напрямок переміщення: > (праворуч), < (ліворуч) або . (на місці);
3. Новий стан автомата.

2. Система команд машини Тюринга

Програма для машини Тюринга становить таблицю, в кожній клітинці якої записана команда (рис. 5.1).

	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	1 ← Q1	0 ← Q2	1 ← Q3	0 → Q6	_ → Q5	0 → Q6	
1	0 ← Q2	1 ← Q2	0 ← Q4	1 → Q6	1 → Q6	1 → Q6	_ → Q7
–	_ → Q7	– ← Q3			– → Q6	– → Q6	
␣			_ → Q6	_ → Q5		_ ← Q1	_

Рис. 5.1. **Приклад програми для машини Тюринга**

Команда дає вказівку: куди пересунути голівку читання / запису, який символ записати в поточну комірку, в який стан перейти машині.

3. Тренажер "Машина Тюринга"

Програму-тренажер "машина Тюринга" можна завантажити за посиланням на сайті ПНС навчальної дисципліни або використати будь-який інший онлайн-тренажер.

Тренажер "Машина Тюринга" – це навчальна модель універсального виконавця (абстрактної обчислювальної машини), запропонованого в 1936 році А. Тюрингом для уточнення поняття алгоритму. Згідно з тезою А. Тюринга, будь-який алгоритм може бути записаний у вигляді програми для машини Тюринга. Доведено, що машина Тюринга за своїми можливостями еквівалентна машині Поста і нормальним алгоритмам Маркова (рис. 5.2).

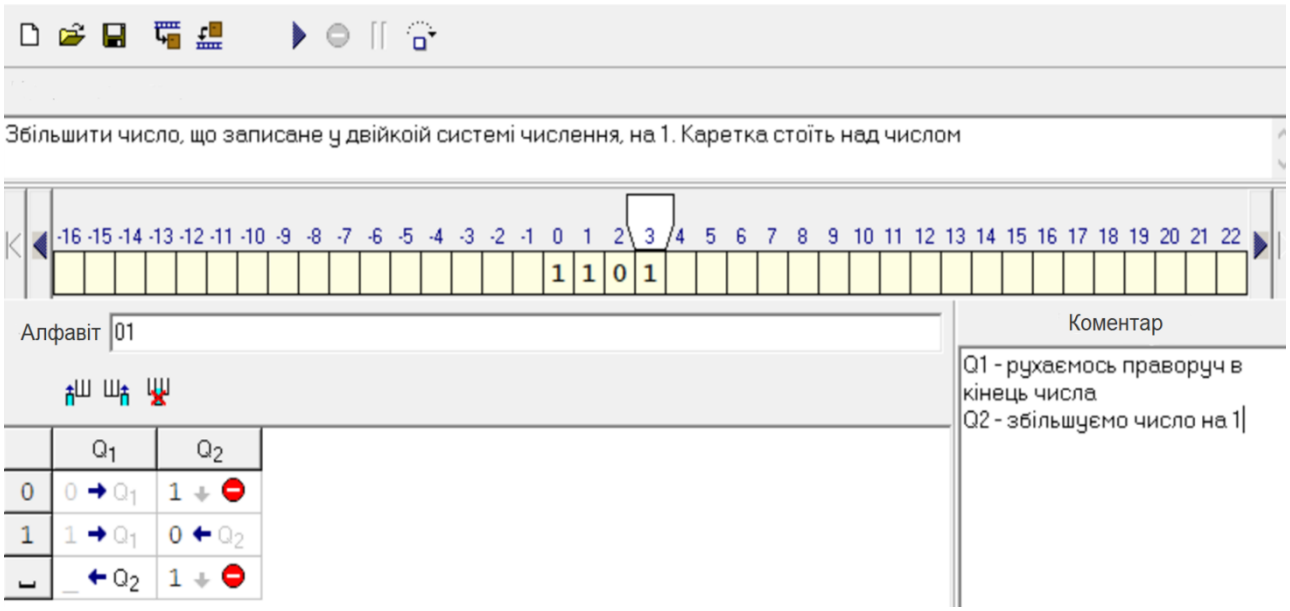


Рис. 5.2. **Тренажер "Машина Тюринга"**

4. Порядок роботи з тренажером "Машина Тюринга"

У верхній частині програми знаходиться поле редактора, в яке можна ввести умову задачі у вільній формі.

Стрічка переміщується вліво і вправо за допомогою кнопок, розташованих ліворуч і праворуч від неї. Подвійним клацанням на комірці стрічки (або клацанням правою кнопкою мишки) можна змінити її вміст.

За допомогою меню "Стрічка" можна запам'ятати стан стрічки у внутрішньому буфері і відновити стрічку з буфера.

У поле "Алфавіт" задають символи обраного алфавіту. Пробіл додається до введених символах автоматично.

У таблиці в нижній частині вікна набирають програму. У першому стовпці записані символи алфавіту, він заповнюється автоматично. У першому рядку перераховують усі можливі стани. Додати і видалити

стовпці таблиці (стану) можна за допомогою кнопок, розташованих над таблицею.

Під час введення команди в клітинку таблиці спочатку потрібно ввести новий символ, потім напрямок переходу (використовуючи символи > або < для переходу вправо або вліво або поставити крапку) і номер стану. Якщо символ пропущений, то за замовчуванням він не змінюється. Якщо пропущений номер стану, то за замовчуванням стан автомата не змінюється.

Праворуч в полі "Коментар" можна вводити в довільній формі коментарі до вирішення. Найчастіше там пояснюють, що означає кожний стан машини Тюринга.

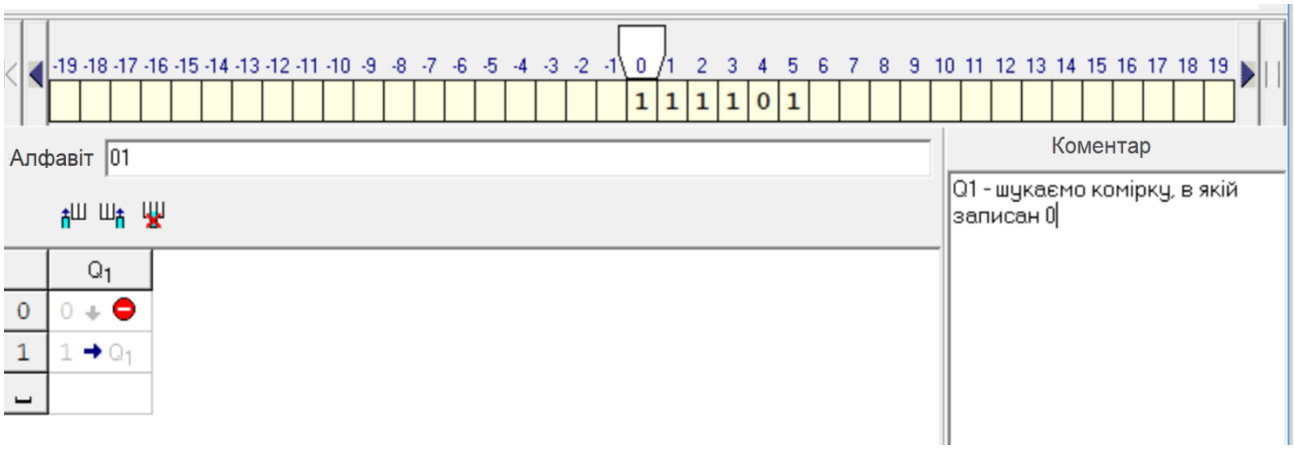
Програма може виконуватися безперервно (F9) або за етапами (F8). Команда, яка зараз буде виконуватися, підсвічується зеленим фоном. Швидкість виконання регулюють за допомогою меню "Швидкість".

Завдання для машини Тюринга можна зберігати в файлах. Зберігається умова завдання, алфавіт, програма, коментарі і початковий стан стрічки. Під час завантаження завдання з файла і збереження в файлі стан стрічки автоматично записується в буфер.

5. Приклади розв'язання задач

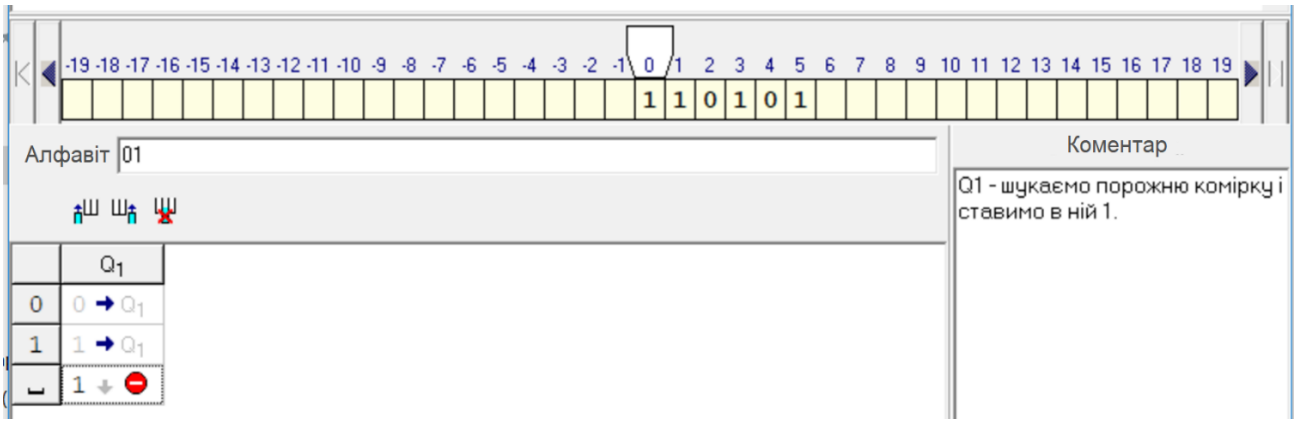
Приклад 1. Каретка зсувається праворуч до тих пір, поки не знайде комірку, в яку записаний нуль. Машина зупиняється над цією коміркою.

Рішення:



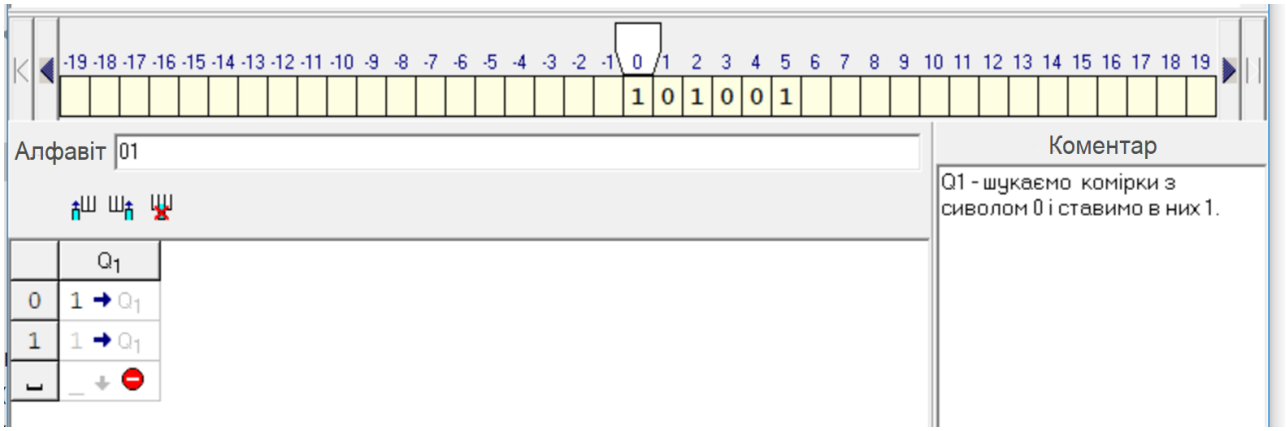
Приклад 2. Поставити в першу вільну позицію символ 1.

Рішення:



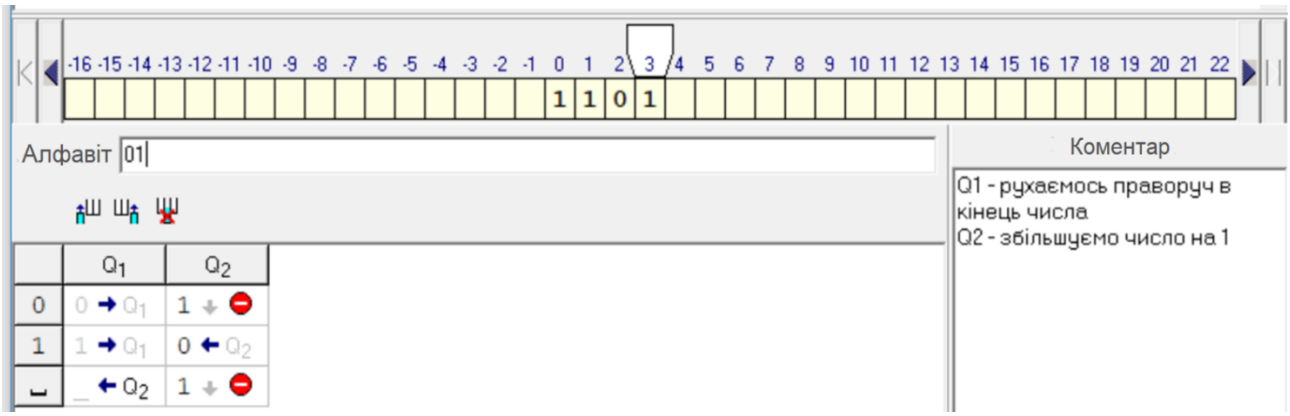
Приклад 3. Замінити всі символи 0 на символ 1.

Рішення:



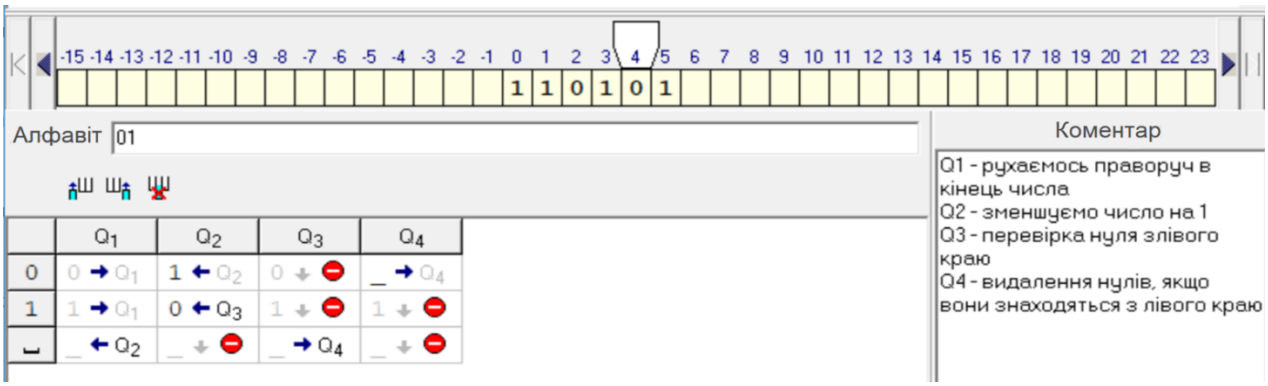
Приклад 4. Збільшити число, записане в двійковій системі числення, на 1. Каретка стоїть над числом.

Рішення:



Приклад 5. Зменшити число, записане в двійковій системі числення, на 1. Каретка стоїть над числом.

Рішення:



Практична частина

Розроблення програм для машини Тюринга

Система оцінювання. За правильно виконане перше завдання студент отримує 4 бали; за правильно виконані перше та друге завдання – 6 балів; якщо правильно виконано всі три завдання – 7 балів.

Приклад.

Описати функціональною таблицею машину Тюринга, що реалізує алгоритм заданий варіантом. Початкова та кінцева конфігурації стандартні. Навести послідовність конфігурацій розробленої машини Тюринга для заданого слова.

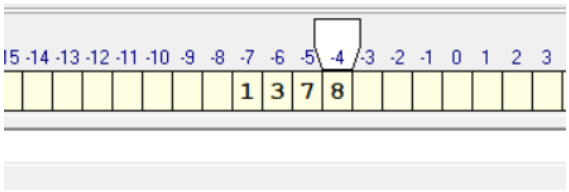
Збільшити на 3 додатне число у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Записати послідовність конфігурацій машини Тюринга для слова 197.

Функціональна таблиця розробленої машини Тюринга має вигляд:

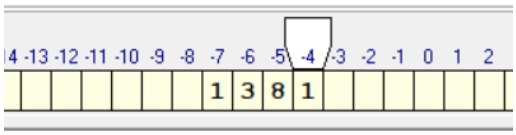
	Q_1	Q_2	Q_3
1	4 Q3	2 → Q3	1 → Q3
2	5 Q3	3 → Q3	2 → Q3
3	6 Q3	4 → Q3	3 → Q3
4	7 Q3	5 → Q3	4 → Q3
5	8 Q3	6 → Q3	5 → Q3
6	9 Q3	7 → Q3	6 → Q3
7	0 ← Q2	8 → Q3	7 → Q3
8	1 ← Q2	9 → Q3	8 → Q3
9	2 ← Q2	0 ← Q2	9 → Q3
0	3 Q3	1 → Q3	0 → Q3
␣		1 → Q3	_ ←

Тестовий приклад – застосування машини Тюринга до слова 1 378:

Початковий стан



Кінцевий стан



Послідовність конфігурацій машини Тюринга для слова 197:

$19q_17 \rightarrow 1q_190 \rightarrow q_2100 \rightarrow 2q_300 \rightarrow 20q_30 \rightarrow 200q_3 \rightarrow 20q_00.$

Завдання 1. Описати машину Тюринга функціональною таблицею, реалізуючи алгоритм, заданий варіантом. Початкова та кінцева конфігурації стандартні. Програмно перевірити модель алгоритму на множині тестових прикладів. Навести послідовність конфігурацій розробленої машини Тюринга для заданого слова.

1. Збільшити на 1 додатне число у системі числення (0, 1, 2, 3). Записати послідовність конфігурацій машини Тюринга для слова 123 333.
2. Збільшити на 1 додатне число у системі числення (0, 1, 2). Записати послідовність конфігурацій машини Тюринга для слова 1 022 222.
3. Збільшити на 1 додатне число у системі числення (0, 1, 2, 3, 4, 5, 6). Записати послідовність конфігурацій машини Тюринга для слова 30 566.
4. Збільшити на 2 додатне число у системі числення (0, 1, 2, 3, 4, 5, 6, 7). Записати послідовність конфігурацій машини Тюринга для слова 7 667.
5. Збільшити на 1 додатне число у системі числення (0, 1, 2, 3, 4). Записати послідовність конфігурацій машини Тюринга для слова 31 444.
6. Збільшити на 1 додатне число у системі числення (0, 1, 2, 3, 4 ,5). Записати послідовність конфігурацій машини Тюринга для слова 31 455.
7. Збільшити на 2 додатне число у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8). Записати послідовність конфігурацій машини Тюринга для слова 7 488.

8. Збільшити на 1 додатне число у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A). Навести послідовність конфігурацій розробленої машини Тюринга для слова 9AA.

9. Збільшити на 1 додатне число у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B). Записати послідовність конфігурацій машини Тюринга для слова ABB.

10. Збільшити на 2 додатне число у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C). Записати послідовність конфігурацій розробленої машини Тюринга для слова BBC.

11. Збільшити на 1 додатне число у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D). Записати послідовність конфігурацій машини Тюринга для слова BDD.

12. Збільшити на 1 додатне число у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E). Записати послідовність конфігурацій машини Тюринга для слова AEE.

13. Збільшити на 2 додатне число у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Записати послідовність конфігурацій машини Тюринга для слова 1EF.

14. Зменшити на 1 число α ($|\alpha| \geq 1$) у системі числення (0, 1, 2, 3, 4, 5, 6). Записати послідовність конфігурацій машини Тюринга для слова 10 000.

15. Зменшити на 1 число α ($|\alpha| \geq 1$) у системі числення (0, 1, 2, 3, 4, 5). Записати послідовність конфігурацій машини Тюринга для слова 2 100 000.

16. Зменшити на 1 число α ($|\alpha| \geq 1$) у системі числення (0, 1, 2, 3, 4). Записати послідовність конфігурацій машини Тюринга для слова 41 000 000.

17. Зменшити на 2 число α ($|\alpha| \geq 2$) у системі числення (0, 1, 2, 3). Записати послідовність конфігурацій розробленої машини Тюринга для слова 111 110 000.

18. Зменшити на 1 число α ($|\alpha| \geq 1$) у системі числення (0, 1, 2, 3, 4, 5, 6, 7). Записати послідовність конфігурацій розробленої машини Тюринга для слова 30 000.

19. Зменшити на 1 число α ($|\alpha| \geq 1$) у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8). Записати послідовність конфігурацій машини Тюринга для слова 7 000.

20. Зменшити на 2 число α ($|\alpha| \geq 2$) у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Записати послідовність конфігурацій машини Тюринга для слова 1 000.

21. Зменшити на 1 число α ($|\alpha| \geq 1$) у системі числення (0, 1, 2). Записати послідовність конфігурацій машини Тюринга для слова 210 000.

22. Зменшити на 2 число α ($|\alpha| \geq 2$) у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E,). Записати послідовність конфігурацій розробленої машини Тюринга для слова 1 010.

23. Зменшити на 1 число α ($|\alpha| \geq 1$) у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D). Записати послідовність конфігурацій машини Тюринга для слова 1 000.

24. Зменшити на 2 число α ($|\alpha| \geq 2$) у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B). Записати послідовність конфігурацій розробленої машини Тюринга для слова 1 110.

25. Зменшити на 1 число у системі числення (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C). Записати послідовність конфігурацій машини Тюринга для слова 1 000.

Завдання 2. Скласти програму для машини Тюринга (за варіантами).

1. Реалізувати функцію – вибір максимального з двох чисел *Max(x, y)* над числами в унарному коді.

2. Реалізувати функцію – вибір мінімального з двох чисел *Min(x, y)* над числами в унарному коді.

3. Реалізувати підрахунок у десятковій системі числення кількості міток послідовності, що розміщується на стрічці.

4. Реалізувати програму для машини Тюринга, яка в послідовності дужок, що відкриваються і закриваються, замінює символом "пробіл" усі пари взаємних дужок, що розташовані поруч. У початковий момент каретка знаходиться навпроти найлівішого символу слова. Наприклад, дано: ")(()(()(", треба отримати: ")(((".

5. Реалізувати функцію обчислення $x - y$ над числами в унарному коді.

6. Реалізувати обчислення предикату $X > Y$ в унарному коді. Якщо умова виконується, то поставити на стрічці мітку, а якщо ні – видалити все.

7. Реалізувати обчислення предикату $X = Y$ *у* двійковому коді. Якщо умова виконується, то поставити на стрічці одиницю, а якщо ні – нуль.

8. Реалізувати обчислення результату цілочислового ділення числа в десятковій системі на 2.

9. Реалізувати обчислення результату цілочислового ділення числа в десятковій системі на 3.

10. Реалізувати обчислення результату залишку від ділення числа в десятковій системі на 2.

11. Реалізувати обчислення результату залишку від ділення числа в десятковій системі на 3.

12. Реалізувати обчислення суми цифр цілого додатного числа, записаного в десятковій формі.

13. Реалізувати обчислення добутку цілого додатного числа записаного в десятковій формі на 2.

14. Реалізувати обчислення кількості нулів у послідовності, складеної із символів двійкового алфавіту.

15. Реалізувати алгоритм в алфавіті $A = \{0, 1\}$, що міняє місцями першу та останню букви слова.

16. Реалізувати алгоритм, що впорядковує цифри додатного цілого числа у системі числення (0, 1, 2) за неспаданням їхніх значень.

17. Реалізувати алгоритм над алфавітом $A = \{0, 1\}$, що міняє місцями перший нуль і останню одиницю.

18. Реалізувати операцію копіювання в алфавіті {*0, 1*}, тобто одержати зі слова α слово α × α.

19. Реалізувати алгоритм над алфавітом $A = \{0, 1\}$, що видає одиницю, якщо у вихідному слові тільки парні нулі і нуль у протилежному випадку.

20. Реалізувати алгоритм в алфавіті $A = \{0, 1\}$, що переставляє букви в слові α так, щоб спочатку йшли всі нулі, а потім – одиниці.

21. Реалізувати алгоритм, що реалізує функцію циклічне зрушення двійкового числа на одну комірку.

22. Реалізувати алгоритм в алфавіті $A = \{1, 2, 3\}$, що аналізує послідовність цифр у слові й видає "+", якщо цифри утворять неспадну послідовність, і "–" у протилежному випадку.

23. Реалізувати виділення підрядка, який розташовано на стрічці між двома символами $*$ (перша пара) в алфавіті $\{a, b, *\}$. Якщо послідовність $"* \ldots *"$ відсутня на стрічці, то стерти все.

24. У слові α в алфавіті $\{a, b\}$ стерти все, окрім $\beta = aabb$. Якщо такої послідовності не існує, то стерти все.

25. Реалізувати алгоритм над алфавітом $\{a, b\}$, що становить букви у зворотному порядку.

Завдання 3. Скласти програму для машини Тюринга.

Перепишіть цифри числа, що задано у вісімковій системі числення, у порядку спадання (не зростання) їхніх значень.

Контрольні запитання до лабораторної роботи 5

1. Які стани каретки машини Тюринга припустимі в одному такті роботи програми?
2. Що входить до складу машини Тюринга?
3. Коли припиняється робота машини Тюринга?
4. Яким чином позначають завершений стан машини Тюринга?
5. Що становить команда машини Тюринга?

Лабораторна робота 6
Розроблення нормальних алгоритмів Маркова

Мета роботи:

- вивчити нормальні алгоритми Маркова та способи їх розроблення;
- отримати навички з розроблення нормальних алгоритмів Маркова.

Теоретична частина

1. Нормальні алгоритми Маркова

За змістом нормальні алгоритми Маркова близькі до ідей Тюринга, однак, в них не використовують уявлення про будь-які машини. Алгоритм задають системою підстановок, які вказують, які заміни символів необхідно зробити і в якому порядку ці підстановки повинні слідувати. Такий підхід був запропонований Марковим А. А. На початку 50-х років XX ст. було введено поняття нормального алгоритму (сам Марков називав їх "алгорифм").

Знову слід розглянути деякий алфавіт А, що містить кінцеве число знаків (літер) і ввести ряд визначень:

Слово – *це будь-яка кінцева послідовність знаків алфавіту.*

Число символів у слові називають його довжиною.

Слово, довжина якого дорівнює нулю, називають порожнім.

Слово s називають підсловом слова q, якщо q можна подати у вигляді q = rst, де r і t- будь-яке слово в тому ж алфавіті (у тому числі і порожні).

Тепер можна визначити поняття алгоритму (яке не є строгим):

Алгоритмом *в алфавіті A називають ефективно обчислювану функцію, областю визначення якої слугує якась підмножина безлічі всіх слів в алфавіті A і значеннями якої також є слова в алфавіті A.*

В алгоритмах Маркова в якості елементарного етапу алгоритму приймають підстановку одного слова замість іншого. Нехай в алфавіті *A* побудовано вихідне слово *P*, яке містить підслово *Pr* (у загальному випадку таких підслів у вихідному слові може бути кілька), а також є деяке слово *Pk* у тому ж алфавіті.

Підстановкою *називають заміну першого по порядку підслова Pr вихідного слова P на слово Pk. Позначають підстановку Pr → Pk.*

Алгоритм у цій формі подання задають системою підстановок, яка становить послідовність (список) підстановок. Якщо в цьому списку є підстановки з лівими частинами, які входять в P, то першу з них застосовують до P, у результаті чого воно переходить в інше слово P_1. До нього знову застосовують схему підстановок і т. д. Процес припиняється в двох випадках: або в списку не знайшлося підстановки з лівою частиною, що входить в P_n, або під час отримання P_n було застосовано останню підстановку.

2. Побудова нормальних алгоритмів Маркова

Нормальний алгоритм (рис. 6.1) задає метод перетворення рядків за допомогою системи підстановок. Кожна підстановка складається з слова-зразка і слова-заміни, розділених символом →. На кожному етапі заміни підстановки переглядають по порядку зверху вниз, і виконують першу з них, яка підійшла: перше знайдене слово-зразок робочого рядка замінюють на слово-заміну.

Зразок		Заміна
1A	→	A1
1*1	→	A*
1*	→	*B
B	→	1
A	→	C
C	→	1
*	→	
	→	.
	→	

Рис. 6.1. **Приклад нормальних алгоритмів Маркова**

Слова ліворуч і праворуч від знака → можуть бути (а можуть і не бути) укладені в апострофи або подвійні лапки. Наступні підстановки рівносильні і визначають заміну літери "а" на поєднання "бв":

а → бв;

'а' → 'бв';

"а" → "бв";

'а' → "бв".

Ліва частина (слово-зразок) може бути відсутньою, в цьому випадку слово-заміна ставлять у самий початок робочого слова. Зазвичай така заміна повинна стояти останньою в списку замін (інакше відбувається зациклення).

Права частина підстановки теж може бути відсутньою (під час видалення зразка).

Символ "." Після слова-заміни позначає термінальну (кінцеву) підстановку, після якої виконання алгоритму закінчується. Наприклад:

'А' →'б'. – замінити "а" на "б" і зупинити програму;

* →. – стерти знак "*" і зупинити програму.

3. Тренажер "Нормальні алгоритми Маркова"

Програму-тренажер "Нормальні алгоритми Маркова" можна завантажити за посиланням ПНС навчальної дисципліни або використати будь-який інший онлайн-тренажер.

Тренажер "Нормальні алгоритми Маркова" – це навчальна модель універсального виконавця, запропонованого в 1940-х роках XX ст. Марковим А. А. для уточнення поняття алгоритму. Марков припустив, що будь-який алгоритм може бути записаний у вигляді нормального "алгорифму" (учений вважав, що правильно вимовляти це іноземне слово саме так). Пізніше було доведено, що нормальні алгоритми Маркова еквівалентні за своїми можливостями іншим універсальним виконавцям: машині Тюринга і машині Поста.

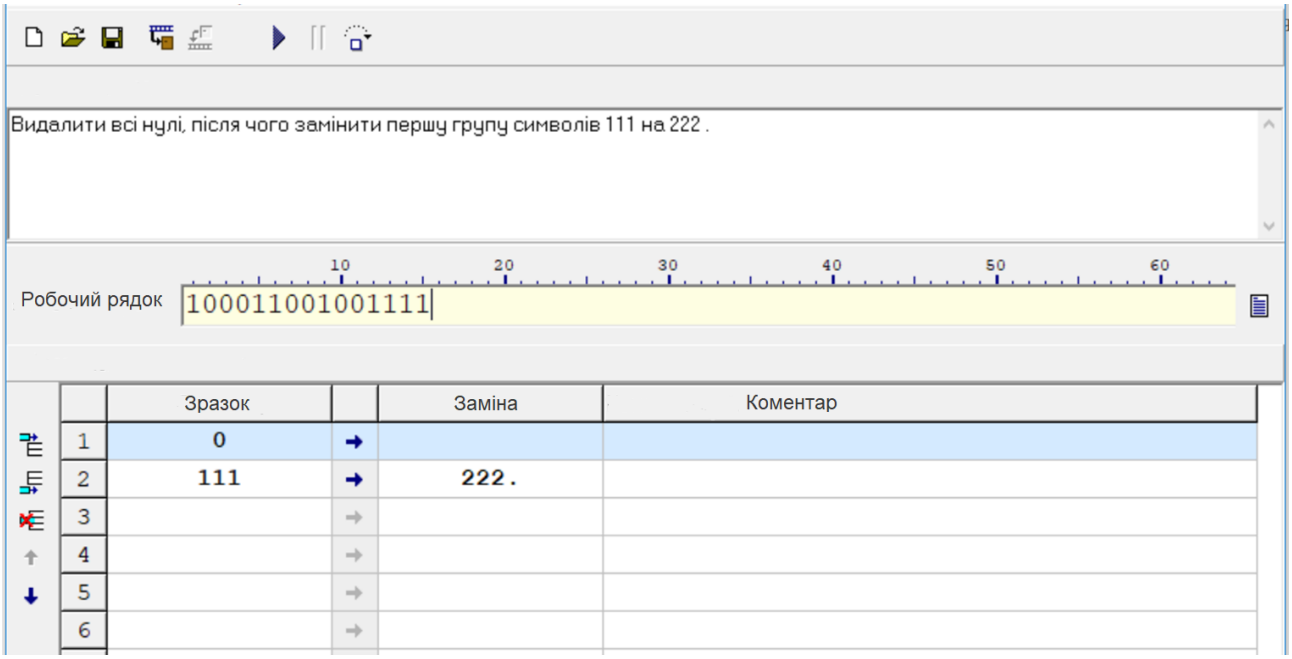


Рис. 6.2. **Тренажер "Нормальні алгоритми Маркова"**

4. Порядок роботи з тренажером "Нормальні алгоритми Маркова"

У верхній частині програми знаходиться поле редактора, в яке можна ввести умову завдання у вільній формі.

Систему підстановок, що задає нормальний алгоритм Маркова, набирають у вигляді таблиці в нижній частині вікна програми.

Програма може виконуватися безперервно (F9) або по етапах (F8). Команда, яка зараз буде виконуватися, підсвічується зеленим фоном. Швидкість виконання регулюють за допомогою меню "Швидкість".

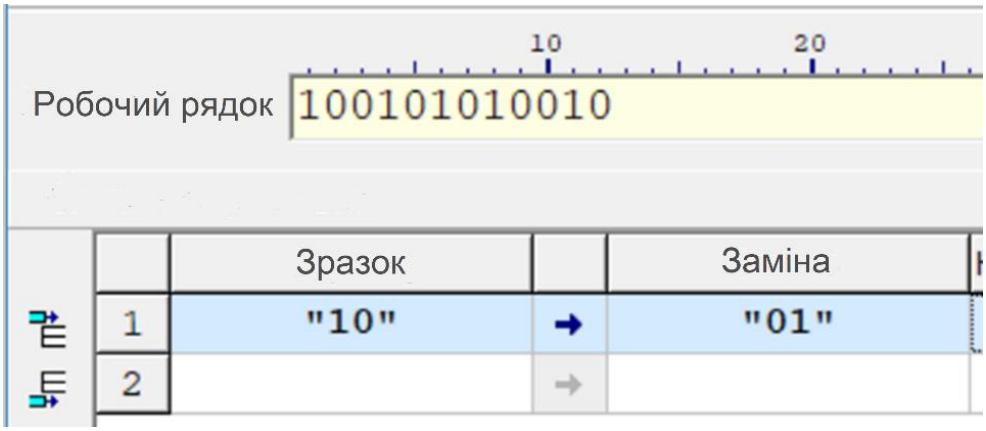
Завдання можна зберігати в файлах і завантажувати з файлів. Зберігається умова завдання, вихідне слово і система підстановок.

Протокол роботи алгоритму, в якому показані всі послідовні заміни, викликають натисканням клавіш Ctrl + P.

Приклади вирішення завдань

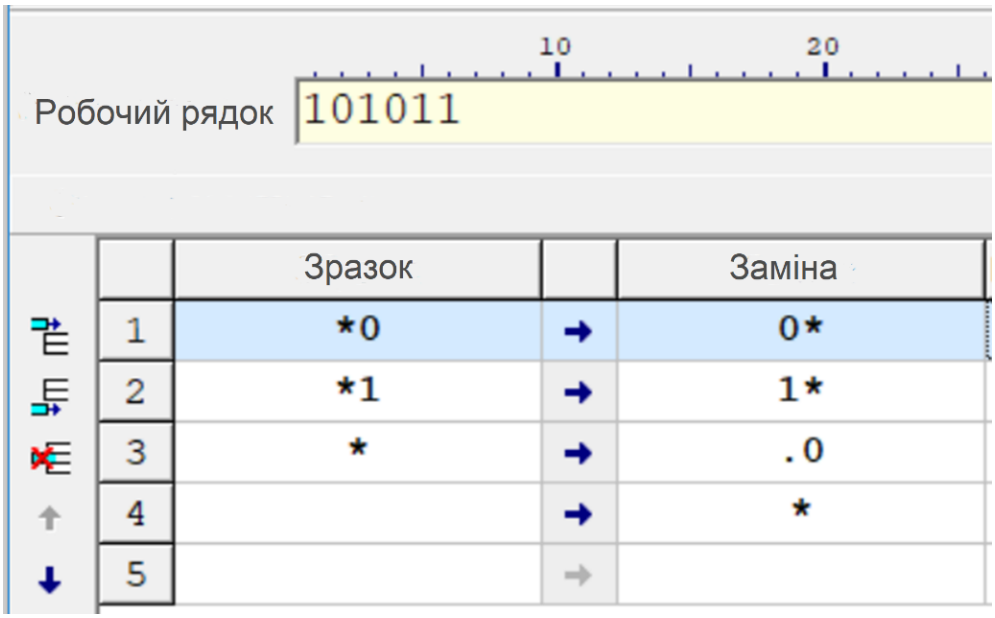
Приклад 1. Дано слово, що складається з цифр 0 і 1. Побудувати нормальний алгоритм Маркова, який цифри 0 переносить вліво, цифри 1 – вправо.

Рішення:



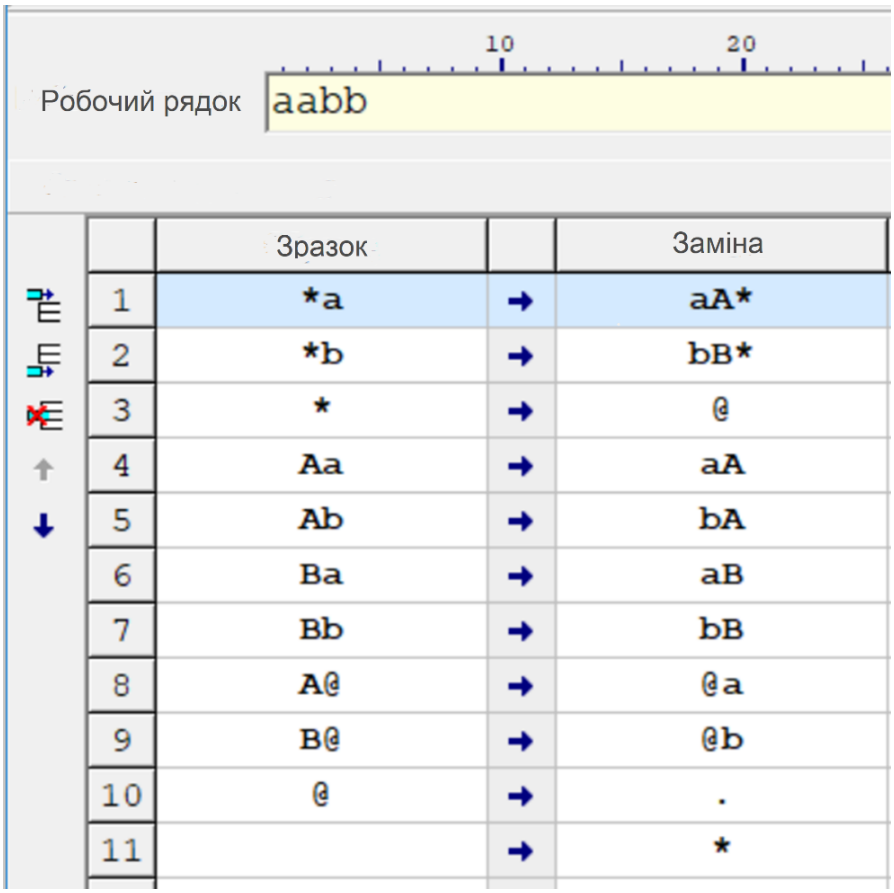
Приклад 2. Помножити двійкове число на 2, приписавши до нього 0 в кінці.

Рішення:



Приклад 3. Подвоїти слово, приписавши до нього в кінець копію.

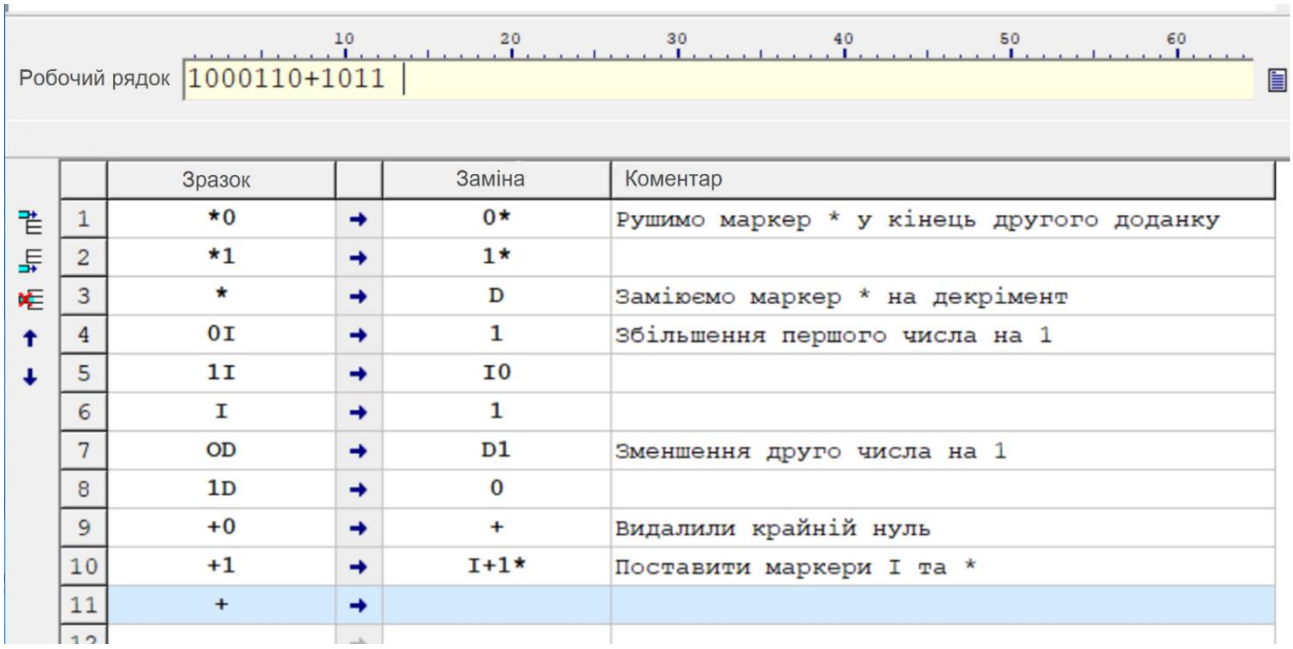
Рішення:



	Зразок		Заміна
1	*a	→	aA*
2	*b	→	bB*
3	*	→	@
4	Aa	→	aA
5	Ab	→	bA
6	Ba	→	aB
7	Bb	→	bB
8	A@	→	@a
9	B@	→	@b
10	@	→	.
11		→	*

Приклад 4. Додати два двійкових числа шляхом послідовного збільшення одного на 1 і зменшення другого на 1.

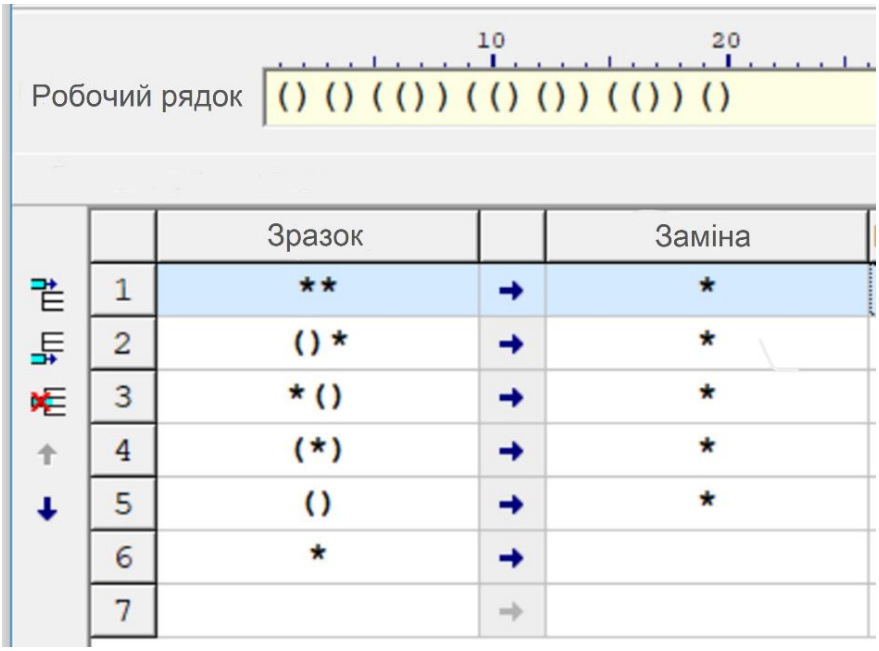
Рішення:



	Зразок		Заміна	Коментар
1	*0	→	0*	Рушимо маркер * у кінець другого доданку
2	*1	→	1*	
3	*	→	D	Заміюємо маркер * на декримент
4	0I	→	1	Збільшення першого числа на 1
5	1I	→	I0	
6	I	→	1	
7	0D	→	D1	Зменшення друго числа на 1
8	1D	→	0	
9	+0	→	+	Видалили крайній нуль
10	+1	→	I+1*	Поставити маркери I та *
11	+	→		

Приклад 5. Дано послідовність дужок. За допомогою нормальної системи підстановок Маркова визначити правильність структури, складеної з відкриваючих та закриваючих дужок. Якщо все правильно, то рядок повинен бути порожнім, а якщо вираз неправильний, то повинні залишитися "неправильні" дужки.

Рішення:



Практична частина

Розроблення програм для машини Тюринга

Система оцінювання. За правильно виконане перше завдання студент отримує 4 бали; за правильно виконані перше та друге завдання – 6 балів; якщо правильно виконано всі три завдання – 7 балів.

Приклад.

Скласти абстрактну модель алгоритму у вигляді нормального алгоритму Маркова над алфавітом А. На конкретних прикладах вхідних слів продемонструвати роботу розробленого алгоритму.

Збільшити на 1 додатне ціле число у системі числення (алфавіті) А (0, 1, 2).

Таблиця (схема) нормальних підстановок Маркова має вигляд:

1	*2	→	2*	
2	*1	→	1*	пересування маркеру в кінець
3	*0	→	0*	
4	*	→	i	
5	0i	→	.1	додавання одиниці до останньої цифри слова
6	2i	→	i0	
7	1i	→	.2	
8	i	→	.1	випадок переповнення розряду
9		→	*	ставимо маркер

Тестовий приклад 1 – застосування алгоритму до слова 12 020 122:

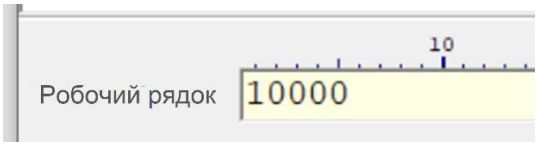
Початковий стан:

Кінцевий стан:

Тестовий приклад 2 – застосування алгоритму до слова 2 222:

Початковий стан:

Кінцевий стан:



Завдання 1. Скласти абстрактну модель алгоритму у вигляді нормального алгоритму Маркова над алфавітом А. На конкретних прикладах вхідних слів продемонструвати роботу розробленого алгоритму.

1. Збільшити на 2 додатне ціле число у системі числення А (0, 1, 2, 3, 4).

2. Зменшити на 2 додатне ціле число у системі числення А (0, 1, 2, 3, 4).

3. Збільшити на 1 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6, 7).

4. Зменшити на 3 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6, 7).

5. Збільшити на 3 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5).

6. Зменшити на 2 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5).

7. Збільшити на 4 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6, 7, 8).

8. Зменшити на 3 додатне число ціле у системі числення А (0, 1, 2, 3, 4, 5, 6, 7, 8).

9. Збільшити на 3 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

10. Зменшити на 4 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

11. Збільшити на 2 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, А).

12. Зменшити на 3 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, А).

13. Збільшити на 3 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6, 7 ,8, 9, А, В).

14. Зменшити на 2 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, А, В).

15. Зменшити на 3 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6).

16. Збільшити на 4 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6).

17. Зменшити на 2 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, А, В, С).

18. Збільшити на 2 додатне ціле число у системі числення А (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, А, В, С).

19. Зменшити на 4 додатне ціле число у системі числення A (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D).

20. Збільшити на 5 додатне ціле число у системі числення A (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D).

21. Збільшити на 3 додатне ціле число у системі числення A (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E).

22. Зменшити на 6 додатне ціле число у системі числення A (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E).

23. Збільшити на 3 додатне ціле число у системі числення A (0, 1, 2, 3).

24. Збільшити на 7 додатне ціле число у системі числення A (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

25. Зменшити на 5 додатне ціле число у системі числення A (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

Завдання 2. Скласти модель алгоритму у вигляді нормального алгоритму Маркова над алфавітом A (за варіантами):

1. Задано алфавіт A = {0, 1, 2, 3}. Перетворити слово *p* так, щоб спочатку йшли всі парні цифри (0 і 2), а потім − всі непарні.

2. Задано алфавіт A = {*a* , *b*, *c*}. Перетворити слово *p* так, щоб спочатку йшли всі символи *a*, потім − всі символи *b* і в кінці − всі символи *c*.

3. Задано алфавіт A = {*a*, *b*, *c*}. Побудувати нормальний алгоритм Маркова, який визначає, зі скількох різних символів складено слово *p*; відповідь отримати в одиничній системі числення (наприклад: *acaac* → 11).

4. (а) Задано алфавіт A = {*a*, *b*}. Перенести перший символ непорожнього слова *p* в кінець слова.

(б) Задано алфавіт A = {*a*, *b*}. Перенести останній символ непорожнього слова *p* в початок слова.

5. Задано алфавіт A = {*a*, *b*}. У непорожньому слові *p* переставити перший і останній символи.

6. Реалізувати нормальний алгоритм над алфавітом A = {*a*, *b*}, що переставляє букви у зворотному порядку.

7. Задано алфавіт A = {*a*, *b*}. Нехай слово *p* має непарну довжину. Видалити з нього середній символ.

8. Задано алфавіт A = {a, b}. У слові *p* всі символи *a* замінити на *b*, а всі (колишні) символи *b* – на *a*.

9. Задано алфавіт A = {*a*, *b*, *c*}. Подвоїти кожен символ у слові *p* (наприклад: *bacb* → *bbaaccbb*).

10. Задано алфавіт A = {*a*, *b*}. Приписати справа до слова *p* стільки одиниць, скільки всього символів входить у *p* (наприклад: *babb* → *babb1111*).

11. Задано алфавіт A = {*a*, *b*}. Нехай слово *p* має парну довжину (0, 2, 4…). Видалити праву половину цього слова.

12. Задано алфавіт A = {*a*, *b*}. Нехай довжина слова *p* кратна 3. Видалити праву третину цього слова.

13. Задано алфавіт A = {*a*, *b*}. Приписати справа до слова *p* стільки одиниць, із скількох підряд символів *a* починається це слово (наприклад: *aababa* → *aababa11*).

14. (а) Задано алфавіт A = {*a*, *b*, *c*}. Видалити зі слова *p* друге входження символу *a*, якщо таке є.

(б) Задано алфавіт A = {*a*, *b*, *c*}. Видалити зі слова *p* третє входження символу *a*, якщо таке є.

15. Задано алфавіт A = {*a*, *b*}. Якщо у непорожньому слові *p* співпадають перший і останній символи, то видалити обидва цих символи, а інакше слово не міняти.

16. (а) Задано алфавіт A = {*a*, *b*, *c*}. Залишити в слові *p* тільки перше входження символу *a*, якщо таке є.

(б) Задано алфавіт A = {*a*, *b*, *c*}. У непорожньому слові *p* залишити тільки останній символ.

17. Задано алфавіт A = {*a*, *b*}. Визначити, чи є слово *p* паліндромом. Відповідь: слово *a*, якщо є, або порожнє слово інакше.

18. Задано алфавіт A = {*a*, *b*, *c*}. Зі всіх входжень символу *a* в слові *p* залишити лише її останнє входження, якщо таке є.

19. Задано алфавіт A = {*a*, *b*, *c*}. Якщо слово *p* починається з символу *a*, то замінити *p* на порожнє слово, інакше *p* не міняти.

20. Задано алфавіт A = {*a*, *b*}. Якщо слово *p* містить одночасно символи *a* і *b*, тоді замінити *p* на порожнє слово.

21. Задано алфавіт A = {*a*, *b*, *c*}. Якщо букви в непорожньому слові *p* невпорядковані за абеткою, то замінити *p* на порожнє слово, а інакше *p* не міняти.

22. Задано алфавіт A = {*a*, *b*, *c*}. Якщо *p* відрізняється від слова *abaca*, тоді замінити його на порожнє слово.

23. Задано алфавіт A = {0, 1}. Вважаючи непорожнє слово *p* записом двійкового числа, визначити, чи є це число степенем 2 (1, 2, 4 і т. д.). Відповідь: слово 1, якщо є, або слово 0 інакше.

24. Задано алфавіт A = {0, 1, 2, 3}. Вважаючи непорожнє слово *p* записом четвіркового числа, перевірити, чи воно є парним. Відповідь: слово 0, якщо є парним, і слово 1 інакше.

25. Реалізувати алгоритм над алфавітом A = {0, 1}, що видає одиницю, якщо у вихідному слові тільки парні нулі й нуль у протилежному випадку.

Завдання 3. Розробить алгоритм додавання двох чисел, що задані у шестнадцятковій системі числення та розділені знаком +.

Контрольні запитання до лабораторної роботи 6

1. Які підстановки використовують у нормальних алгоритмах Маркова?
2. Що називають нормальним алгоритмом Маркова?
3. Яким чином задають нормальний алгоритм Маркова?
4. Що означає твердження про еквівалентність формалізмів?
5. Яким символом позначають термінальну підстановку?

Рекомендована література

Основна

1. Козак Л. І. Основи програмування: навчальний посібник / Л. І. Козак, І. В. Костюк, С. П. Стасевич. – Львів : "Новий Світ-2000", 2020. – 328 с.

2. Креневич А. П. Алгоритми і структури даних : підручник / А. П. Креневич. – Київ : ВПЦ "Київський Університет", 2021. – 200 с.

3. Федорченко В. М. Алгоритмізація та програмування [Електронний ресурс] : навч. посіб. / В. М. Федорченко, О. В. Щербаков, Ю. Е. Парфьонов ; кер. проєкту В. М. Анохін. – Харків : ХНЕУ ім. С. Кузнеця, 2016.

Додаткова

4. Бородкіна І. Теорія алгоритмів : посібник для студентів вищих навчальних закладів / І. Бородкіна. – Київ : Центр навчальної літератури, 2019. – 184 с.

5. Коваль В. С. Алгоритми і структури даних : навчальний посібник / В. С. Коваль, П. Р. Струбицький. – Тернопіль : ФОП "Шпак В. Б.", 2017. – 74 с.

6. Кормен Томас Г. Вступ до алгоритмів / Томас Г. Кормен, Чарлз Е. Лейзерсон, Роналд Л. Рівест, Кліфорд Стайн ; пер. з англ. – Київ : К.І.С., 2019. – 1 288 с.

7. Матвієнко М. П. Теорія алгоритмів : навч. посіб. / М. П. Матвієнко. – Київ : ЛіраК, 2019. – 340 с.

Інформаційні ресурси

8. Алгоритми і структури даних для початківців: динамічний масив [Електронний ресурс]. – Режим доступу : https://echo.lviv.ua/dev/7031.

9. Алгоритми і структури даних для початківців: складність алгоритмів [Електронний ресурс]. – Режим доступу : https://echo.lviv.ua/dev/7084.

10. Алгоритми і структури даних для початківців: стеки і черги [Електронний ресурс]. – Режим доступу : https://echo.lviv.ua/dev/6992.

11. Щербаков О. В. Основи алгоритмізації [Електронний ресурс]. / О. В. Щербаков. – Режим доступу : https://pns.hneu.edu.ua/course/view.php?id=4976.

12. Algorithms and Data Structures [Electronic resource]. – Access mode : https://sites.google.com/site/indy256/algo_cpp.

13. Sorting Algorithm Animations [Electronic resource]. – Access mode : http://www.sortingalgorithms.com.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

ОСНОВИ АЛГОРИТМІЗАЦІЇ

Методичні рекомендації до виконання лабораторних робіт

для студентів спеціальності 121 "Інженерія програмного забезпечення" освітньої програми "Інженерія програмного забезпечення" першого (бакалаврського) рівня

Самостійне електронне текстове мережеве видання

Укладачі: **Щербаков** Олександр Всеволодович
Фролов Олег Васильович

Відповідальний за видання *І. О. Ушакова*

Редактор *В. О. Дмитрієва*

Коректор *В. Ю. Труш*

План 2023 р. Поз. № 186 ЕВ. Обсяг 68 с.

Видавець і виготовлювач – ХНЕУ ім. С. Кузнеця, 61166, м. Харків, просп. Науки, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру ***ДК № 4853 від 20.02.2015 р.***