## Протокол

## Лабораторна робота №4. Варіант 14

### *I частина*

### Завдання

1. Розробити програму роботи зі стеком, яка реалізує операції

додавання, видалення елементів зі стеку і відображення поточного стану

стеку. Реалізувати стек: а) масивом; б) списком (на оцінку «добре»).

2. Розробити програму роботи з чергою, яка реалізує операції додавання,

видалення елементів з черги і відображення поточного стану черги.

Реалізувати чергу: а) масивом (використовувати кільцеву чергу); б) списком

(на оцінку «відмінно»).

3. При реалізації стеку масивом забезпечити розміщення двох стеків в

одному масиві. Один стек розміщується на початку масиву і зростає до кінця,

а другий розміщується в кінці масиву і росте до початку. Заповнити і

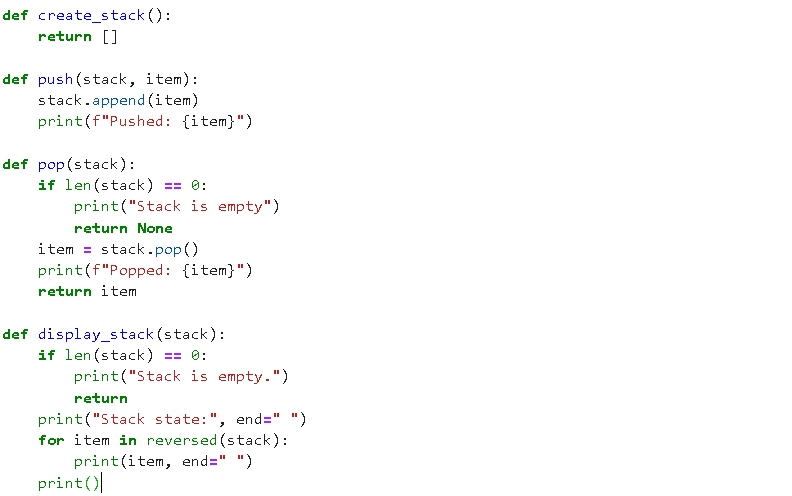
звільнити стеки довільним чином, вводячи значення з клавіатури.

Елементами стеку є  дійсні числа. У вигляді списку реалізувати один стек (на

оцінку «відмінно»).

### Реалізація

**Стек реалізований масивом**



Цей код створює та керує структурою даних, відомою як стек. Стек працює за принципом "останнім прийшов – першим пішов". Починається зі створення порожнього стеку. Додавання елемента (push) кладе новий елемент на вершину стеку, тоді як видалення елемента (pop) забирає елемент з вершини. Якщо спробувати взяти елемент з порожнього стеку, код повідомляє, що стек порожній. Функція відображення показує поточний стан стеку, виводячи всі його елементи.

**Результат:**

****

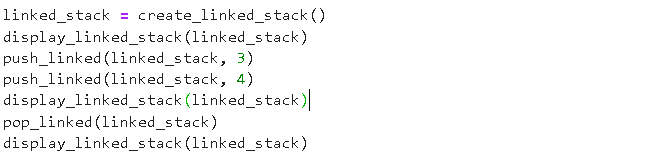
#### 

**Cтек реалізований списком**

****

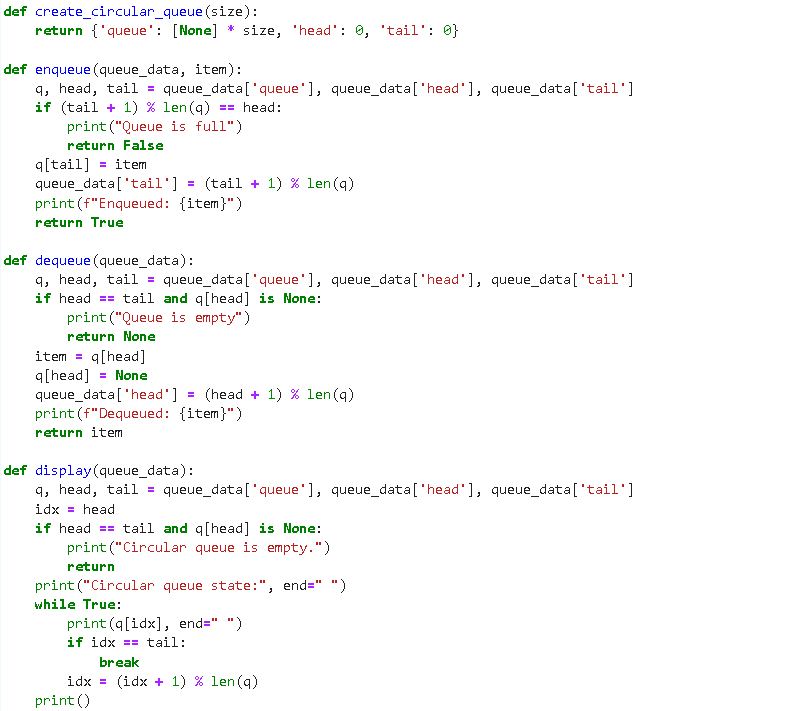
Цей код створює та керує стеком за допомогою зв'язного списку, де кожен елемент (вузол) має посилання на наступний. Він починає з ініціалізації порожнього стеку, де "top" вказує на вершину стеку, а "nodes" зберігає всі вузли. Додавання елемента до стеку створює новий вузол з даними та посиланням на попередній верхній вузол, роблячи новий вузол вершинним. Видалення елемента з стеку видаляє верхній вузол, повертаючи дані та оновлюючи вершину стеку на наступний вузол. Функція відображення проглядає стек від вершини до кінця, показуючи дані кожного вузла.

**Результат:**

****

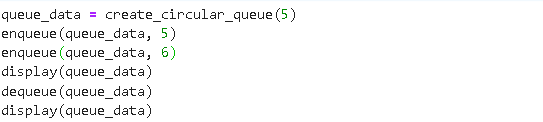
****

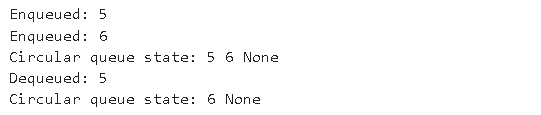
**Кільцева черга за допомогою масиву**

****

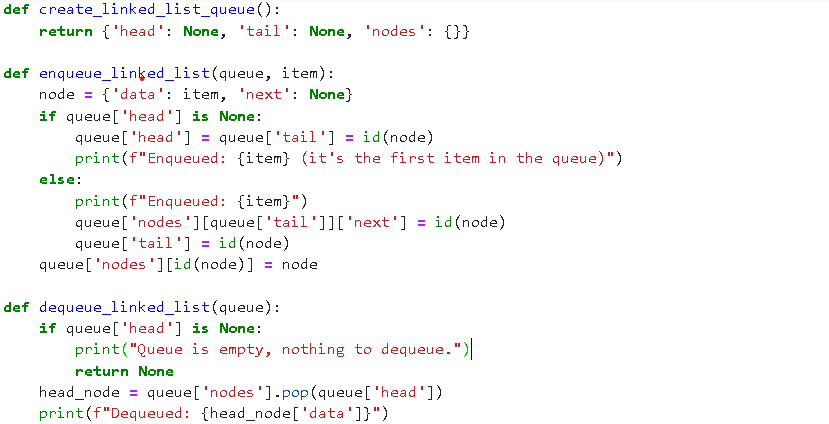
Цей код створює та керує кільцевою чергою, що дозволяє ефективно використовувати обмежений простір масиву для зберігання елементів. Черга ініціалізується з певним розміром, при цьому "head" вказує на початок черги, а "tail" - на кінець. Додавання елемента (enqueue) поміщає новий елемент у кінець черги, якщо вона не заповнена, і переміщає "tail" на наступну позицію. Видалення елемента (dequeue) видаляє елемент з початку черги, оновлюючи "head" на наступну позицію. Якщо черга пуста або заповнена, виводяться відповідні повідомлення. Функція відображення показує поточний стан черги, перебираючи від "head" до "tail". Цей підхід забезпечує циклічне використання масиву, дозволяючи повторно використовувати простір, звільнений від оброблених елементів.

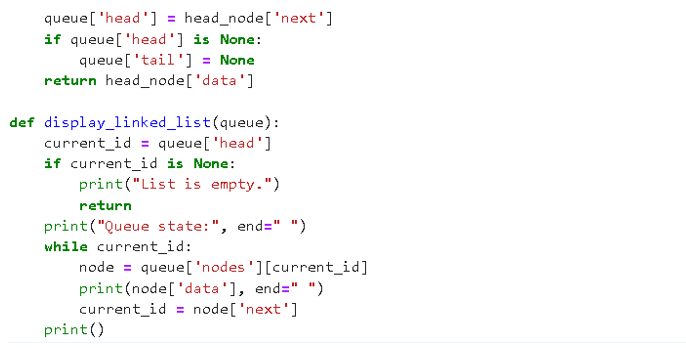
**Результат:**

****

****

**Черга за допомогою зв'язного списку**

****



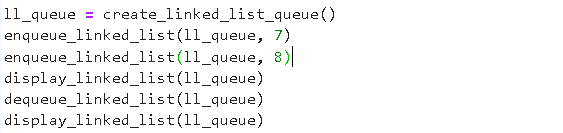
Цей код створює та керує чергою, використовуючи зв'язний список, що дозволяє динамічно додавати та видаляти елементи без обмежень, накладених фіксованим розміром масиву. Кожен новий елемент (вузол) містить дані та посилання на наступний елемент у черзі. Черга має посилання на перший (голова) та останній (хвіст) елементи для швидкого доступу.

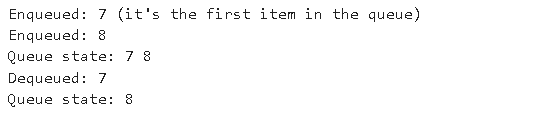
При додаванні елемента (enqueue), створюється новий вузол. Якщо черга порожня, цей вузол стає одночасно і головою, і хвостом черги. Якщо черга не порожня, новий вузол додається в кінець черги, та оновлюються вказівники на хвіст та на наступний вузол попереднього хвоста.

При видаленні елемента (dequeue), видаляється вузол з початку черги. Використовується ідентифікатор голови для знаходження вузла у словнику вузлів, потім цей вузол видаляється зі словника, і голова оновлюється на наступний вузол. Якщо після видалення черга стає порожньою, вказівники на голову та хвіст обнуляються.

Функція відображення (display) проходить через весь зв'язний список від голови до хвоста, виводячи дані кожного вузла, що дозволяє переглянути поточний стан черги.

**Результат:**

****



**Реалізація двох стеків в одному масиві**

****

Цей код організовує два стеки всередині одного спільного масиву. Стеки "ростуть" у протилежних напрямках всередині масиву: перший стек розпочинається з початку масиву і росте в бік кінця, тоді як другий стек починається з кінця масиву і росте в бік початку. Це дозволяє ефективно використовувати весь доступний простір масиву для обох стеків.

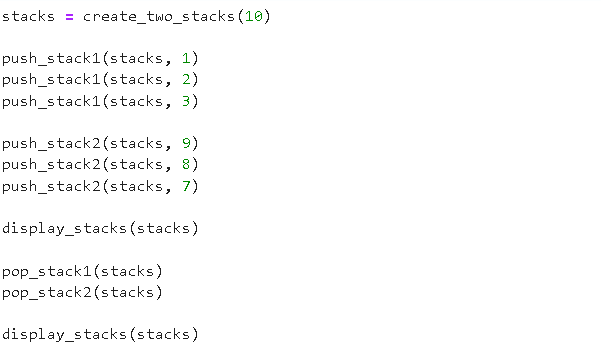
Функція **create\_two\_stacks** ініціалізує структуру даних з масивом заданого розміру та двома індексами, які вказують на вершини стеків (один з них починається з -1, інший — з кінця масиву).

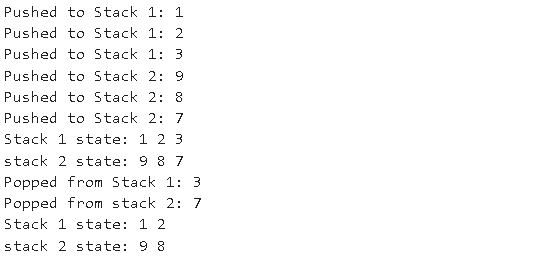
Функції **push\_stack1** та **push\_stack2** додають нові елементи до відповідних стеків. Кожна функція перш ніж додати новий елемент перевіряє, чи не "зіткнулися" стеки всередині масиву, що вказувало б на те, що масив заповнений.

Функції **pop\_stack1** **та pop\_stack2** видаляють елементи з відповідних стеків. Якщо спробувати видалити елемент з порожнього стеку, виводиться повідомлення про помилку.

Функція **display\_stacks** відображає поточний стан обох стеків, показуючи їх елементи.

**Результат:**

****

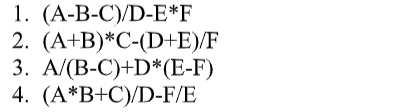
****

### *II частина*

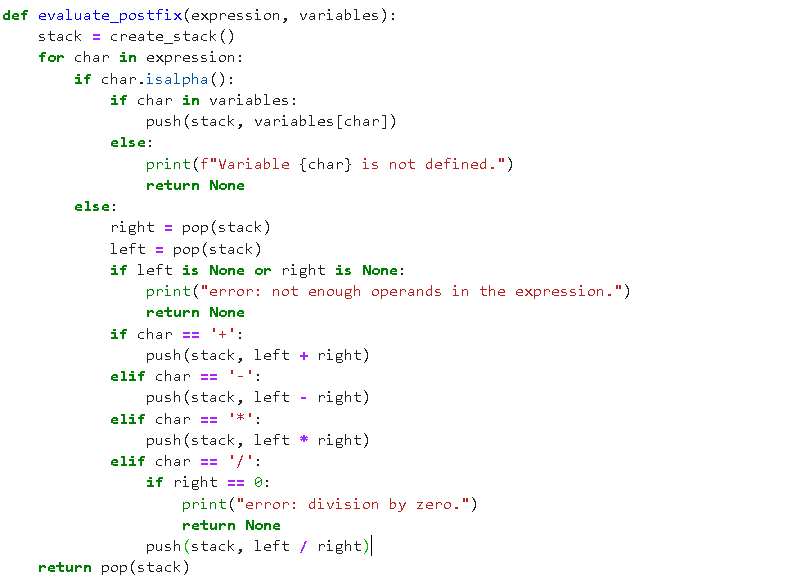
### Завдання

Перетворіть наведені нижче вирази у постфіксну форму і реалізуйте за

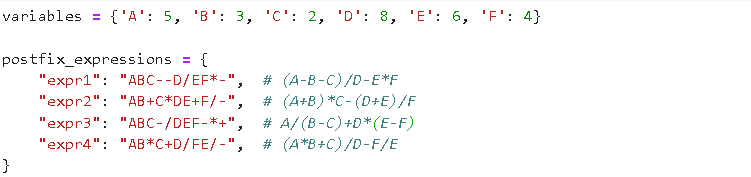
допомогою стеку:



### Реалізація

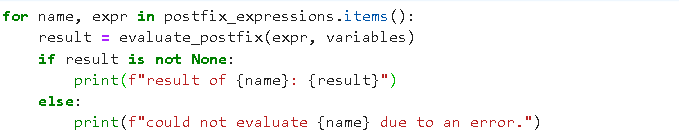


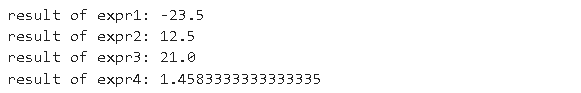
Цей код обчислює математичні вирази, задані у постфіксній формі (тобто оператори йдуть після своїх операндів), використовуючи стек. Для кожного символу у виразі: якщо це буква (змінна), його значення шукається у словнику variables і кладеться на стек; якщо це оператор, зі стеку беруться два верхні елементи (операнди), між якими виконується відповідна операція, а результат знову кладеться на стек. Якщо у виразі зустрічається невизначена змінна або недостатньо операндів для операції, код виводить помилку. У кінці, коли всі символи оброблені, на вершині стеку має залишитися результат обчислення всього виразу, який і повертається як результат функції.



Цей код визначає два словники: один для змінних і їх значень, а другий для математичних виразів у постфіксній формі. У словнику variables кожній змінній (наприклад, 'A', 'B', 'C' тощо) присвоєно конкретне числове значення. У словнику postfix\_expressions зберігаються різні математичні вирази, які необхідно обчислити, і вони представлені таким чином, що оператори йдуть після їх операндів, відповідно до логіки постфіксної нотації. Кожен вираз має унікальний ідентифікатор (наприклад, "expr1", "expr2" тощо) і рядок, що представляє сам вираз.

**Результат:**

****

****