**Стек і Черга**

*Завдання:*

**Для роботи зі стеком розв'язати наступні задачі:**

1. Створити проект, в якому вводиться рядок, що містить символи і числа, який розбивається на два стеки: перший містить символи, другий – числа. Організувати виведення вмісту обох стеків.

2. Створити проект, в якому введений рядок розбивається посимвольно в два стеки: перший містить голосні літери, другий – приголосні. Символи, які не є літерами, ігнорувати. Організувати виведення вмісту обох стеків.

3. Виконати сортування масиву за допомогою стеку.

**Для роботи з чергою:**

* Створити чергу з n елементів (заповнення виконати функцією random).
* Витягти з черги перші k елементів і вивести їх на екран.

Числа ***n*** і ***k*** для кожної операції вводяться користувачем.

1. Створити метод *clear()* для видалення всіх елементів черги. Протестувати роботу даного методу.
2. Напишіть метод для виведення вмісту черги. Вміст черги має виводитися від першого вставленого елементу до останнього. Будьте уважні і простежте за тим, щоб один елемент і вміст порожньої черги виводилися коректно незалежно від положення *head* і *rear*.
3. Зробіть так, щоб була реалізована черга без лічильника елементів. Для цього зробіть копію існуючого проекту. Чи зростає при цьому ефективність методів *insert()* і *remove()*?
4. Створити чергу з пріоритетом.

**Практична частина**

Всі проекти знаходяться на віддаленому репозиторію, силка на який прикріплена до завдання.

Метою цього завдання є дослідити структури даних стек та черга, розробити алгоритми для роботи з ними, протестувати їх функціональність та проаналізувати асимптотичну складність.

**Стек** – лінійний список, у якому всі операції вставки і знищення елементів виконуються лише на одному з кінців списку. **Черга** (одностороння черга) – лінійний список, у якому всі операції вставки здійснюються на одному з кінців списку, а всі операції знищення – на іншому.

**Вирішення завдань**

1. Створити проект, в якому вводиться рядок, що містить символи і числа, який розбивається на два стеки: перший містить символи, другий – числа. Організувати виведення вмісту обох стеків. Код програми знаходиться на віддаленому репозиторію, назва проекту **1\_task\_Stack**.

Програма успішно розподіляє символи рядка, поміщаючи літери в один стек, а числа — в інший. При введенні рядка "abc123def456" результатом є два стеки:

**Стек символів:** f e d c b a

**Стек чисел:** 6 5 4 3 2 1

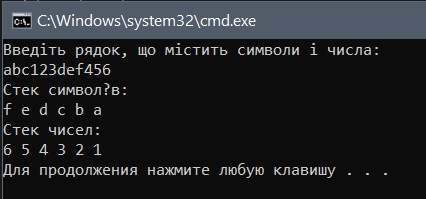


Рис. 1 – Результат роботи першої програми (1\_task\_Stack)

1. Створити проект, в якому введений рядок розбивається посимвольно в два стеки: перший містить голосні літери, другий – приголосні. Символи, які не є літерами, ігнорувати. Організувати виведення вмісту обох стеків. (**2\_tasks\_Stack**)

Програма ефективно розподіляє голосні та приголосні літери по окремих стеках. Для введеного рядка "Hello World!" отримали:

**Стек голосних:** o o e

**Стек приголосних:** d l r W l l H

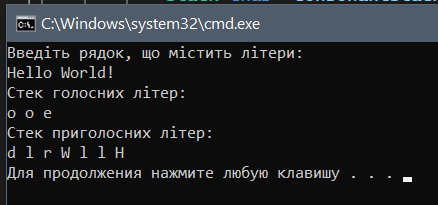


Рис. 2 – Результат роботи другої програми (2\_task\_Stack)

1. Виконати сортування масиву за допомогою стеку. **3\_tasks\_Stack**

Програма сортує масив {34, 3, 31, 98, 92, 23} за допомогою стеків і повертає відсортований масив:

**Відсортований масив:** 3 23 31 34 92 98

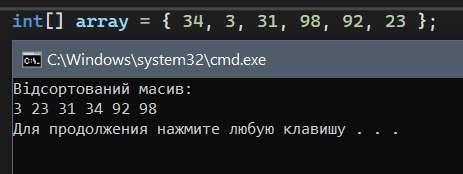


Рис. 3 – Результат роботи третьої програми (3\_task\_Stack)

1. Для виконання роботи з чергою згідно вимог завдання був створений проект з 3 класами які виконують відповідні завдання. **Turn**
   * **Робота з чергою випадкових чисел, видалення та очищення**

При створенні черги з n випадкових чисел та видаленні перших k елементів, програма коректно витягує та відображає ці елементи. Наприклад:

* **Для черги** {7, 36, 1, 59, 26, 23} і k=3 , **перші три елементи**: 7 36 1.
* Функція clear успішно очищує чергу, залишаючи її порожньою.
  + **Реалізація черги без лічильника елементів**

Було розроблено чергу, яка працює без лічильника елементів. Це дозволило уникнути додаткової змінної та покращило ефективність черги, хоча незначно, оскільки операції додавання та видалення все ще мають асимптотичну складність .

* + **Реалізація пріоритетної черги**

Пріоритетна черга реалізована на базі *SortedDictionary*. Наприклад, для елементів із пріоритетами {(10, 1), (20, 3), (30, 2), (40, 1)} отримано таке відображення:

**Вміст пріоритетної черги:**

* + - Елемент: 10, Пріоритет: 1
    - Елемент: 40, Пріоритет: 1
    - Елемент: 30, Пріоритет: 2
    - Елемент: 20, Пріоритет: 3

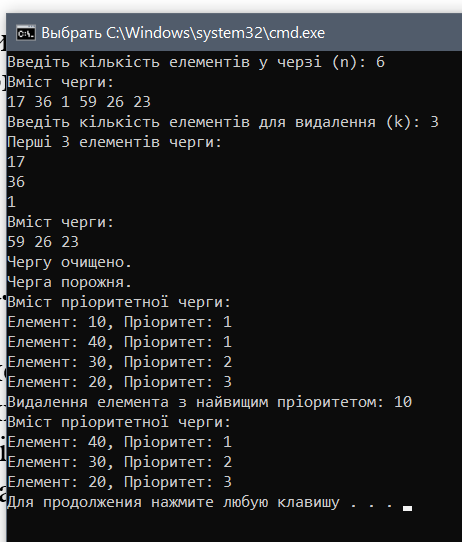


Рис. 4 – Результат роботи четвертої програми (Turn)

**Аналіз асимптотична складность розроблених алгоритмів**

1. **Розподіл символів рядка в два стеки**:
   * Складність алгоритму розподілу символів рядка є , де n – довжина рядка. Прохід по рядку та розподіл по стеках здійснюються за лінійний час.
2. **Сортування масиву за допомогою стека**:
   * Складність цього алгоритму є , оскільки кожен елемент порівнюється з усіма іншими під час вставки. Використання стека як структури сортування тут має обмежену ефективність для великих масивів.
3. **Черга з випадковими числами, видалення та очищення**:
   * Додавання елементів у чергу та їх видалення мають складність . Очищення черги виконується за , де n – кількість елементів у черзі.
4. **Черга без лічильника елементів**:
   * Відсутність лічильника не змінює асимптотичну складність операцій, але спрощує код, виключаючи оновлення лічильника під час вставки/видалення.
5. **Пріоритетна черга**:
   * Операція вставки має складність , оскільки додавання в *SortedDictionary* потребує упорядкування. Видалення елемента з найвищим пріоритетом теж виконується за .

**Види задач, де доцільне використання стеків та черг**

**Стек**:

* + Стек доцільно використовувати в задачах, де необхідне зворотне виведення елементів (наприклад, у реверсі даних, обробці послідовностей викликів, рекурсії).
  + Типові застосування стека включають обробку виразів (обчислення, перевірка балансу дужок) та алгоритми обходу графів.

**Черга**:

* + Черга є доречною у задачах, де важливий порядок обробки елементів (наприклад, черги на обслуговування, потоки даних).
  + Пріоритетні черги використовують, коли елементи мають оброблятись за визначеним пріоритетом (як у плануванні процесів, маршрутизації в мережах).

**Висновок**

Робота зі стеком і чергою дозволяє ефективно вирішувати задачі, де важливий порядок додавання/видалення елементів. При використанні цих структур слід враховувати асимптотичну складність операцій і розмір даних. Стек підходить для задач, де потрібен зворотний порядок обробки, тоді як черга підходить для підтримки прямого порядку обробки або черговості. Пріоритетна черга є корисною для обробки даних за заданим пріоритетом, але вимагає дещо вищої обчислювальної складності для підтримки цього пріоритету.