

# Організація рекурсивних обчислень у Функціональному програмуванні

## Лекція 6

# Рекурсія використовується у двох випадках:

- для опису структур даних, які мають в якості компонент такі ж структури,
- для опису програм, виконанню яких передую виконання їх власних копій.



# Приклади

- Фрактали
- Математичні обчислення на основі рекурентних співвідношень

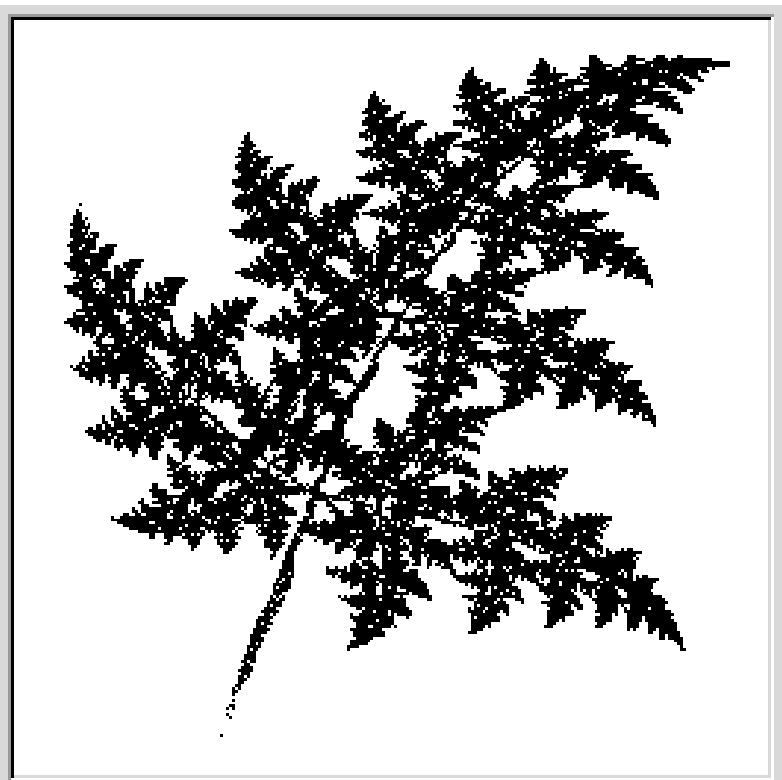
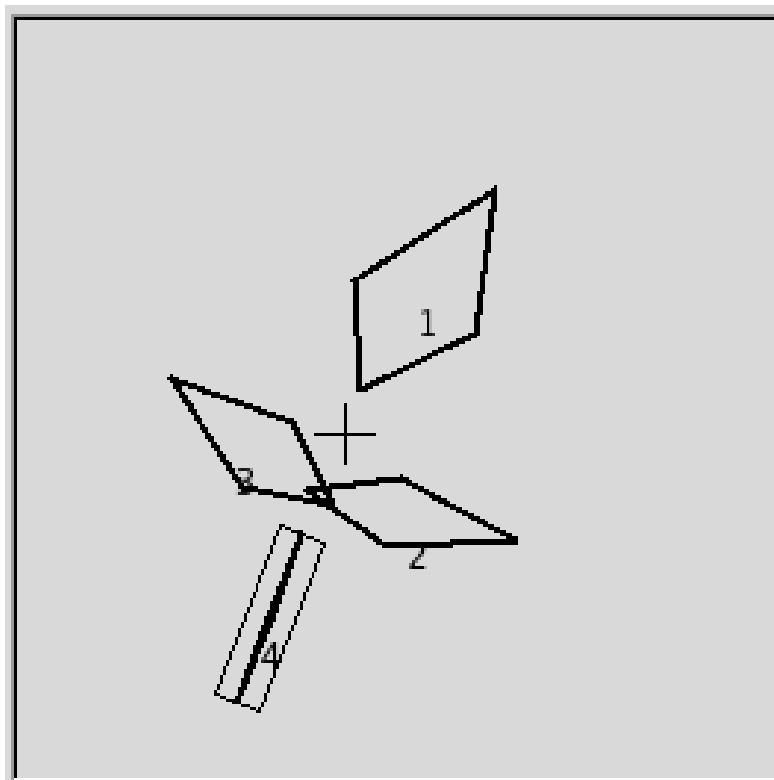
В природі дуже багато рекурсивних об'єктів.






# “Надокучливі” казки


У діда була собака...





Основним способом організації  
повторюваних обчислень у функційному  
програмуванні є **рекурсія**.


Функція є **рекурсивною**, якщо в її  
визначенні міститься виклик цієї самої  
функції.




Рекурсивне визначення складається  
принаймні з двох обчислювальних гілок:  
**термінальної і рекурсивної.**

Термінальних і рекурсивних гілок може  
бути декілька.






Рекурсія у ФП є природнім способом  
організації повторювальних процесів,  
оскільки списки є рекурсивною,  
універсальною формою представлення



**Термінальна гілка** необхідна для закінчення обчислень і містить умову закінчення обчислювального процесу. Без термінальної гілки рекурсивний виклик був би безконечним.


**Термінальна гілка** повертає результат, який є базою для обчислення результатів рекурсивних викликів.

Після кожного виклику функцією самої себе, обчислення повинні наближатися до термінальної гілки.



Для планування **термінальної гілки** рекурсивної функції потрібно вирішити, коли функція може повернути значення без рекурсивного виклику.

Це як правило, найпростіший випадок (наприклад, опрацювання порожнього списку).

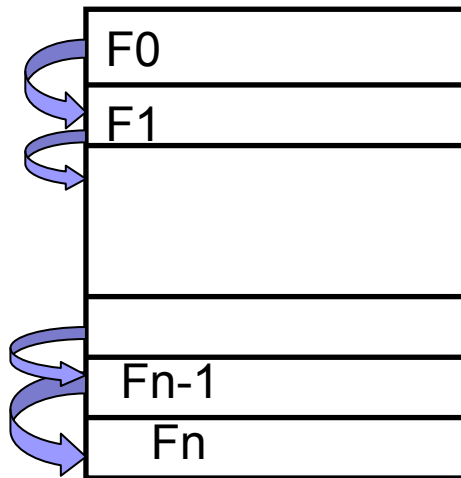


Реалізація рекурсивних викликів функцій  
опирається на механізм стеку викликів.

Адреса повернення і локальні змінні функції  
записуються в стек, кожен наступний  
рекурсивний виклик цієї функції користується  
своїм набором локальних змінних.


При надмірно великій глибині рекурсії може  
наступити переповнення стеку викликів.

Під час рекурсивних обчислень  
утворюється стек перерваних процесів.



Формування стеку перерваних процесів називають **розгорткою рекурсії**.


Вилучення елементів зі стеку перерваних процесів називають **згорткою рекурсії**.



Коли обчислювальний процес доходить до рекурсивної гілки, то поточний процес призупиняється, і новий такий же запускається з початку, але вже на новому рівні (верхній рівень).

Перерваний процес запам'ятовується і перебуває в режимі очікування доти, поки не закінчиться новий.

Новий процес, в свою чергу, може призупинитися і чекати і так далі.



Порядок розміщення термінальних та рекурсивних гілок є важливим.

Спочатку варто розміщувати прості термінальні гілки, далі – складніші (якщо вони є), за ними мають слідувати рекурсивні гілки.

Неправильне розміщення обчислювальних гілок, відсутність термінальних гілок або помилки в означенні термінальних та рекурсивних гілок приводять до некоректних обчислень, нескінченної рекурсії.

# Класифікація рекурсій (стосовно місця виклику)

- Рекурсія за аргументом
- Рекурсія за значенням



# Рекурсія за аргументом

- це якщо у якості результату повертається значення іншої функції, аргумент якої формується на основі рекурсивного виклику .

$$F(x) = G(F(x-1), x)$$

# Рекурсія за значенням

- це якщо рекурсивний виклик є виразом, який формує безпосередньо результат функції.

$$F(x) = F(ax - b/x + 1)$$

# Класифікація рекурсій (стосовно структури обчислювального процесу)

- Проста

- Складна

(паралельна, взаємна/перехресна, вищого порядку)

# Проста рекурсія


- це коли одиничний виклик функції зустрічається в одній чи декількох гілках.

```
(define F (lambda (.....) (.... (F.....) ...  
                                (F....)...)))
```

# Паралельна рекурсія

- це коли тіло визначення даної функції містить виклик іншої функції, декілька аргументів якої є рекурсивними викликами даної функції.

```
(define F (lambda (.....)
              (... (G (F....) ... (F...)...))))
```



Це текстуальна паралельність.  
Тобто не в сенсі часу, а в сенсі логіки.

# Взаємна (перехресна) рекурсія

- це коли у визначенні даної функції викликається інша функція, яка у свою чергу визначена через дану функцію.

```
(define F (lambda (.....)
              (... (G ...) ...)))
(define G (lambda (.....)
              (... (F ..... ) ...)))
```


# Рекурсія вищого порядку

- це коли аргументами рекурсивного виклику функції є рекурсивні виклики тієї ж функції.

Порядок рекурсії визначається рівнем, на якому знаходиться вкладений рекурсивний виклик.


Проста, паралельна, взаємна – це рекурсії нульового порядку.





Якщо функція викликає саму себе, а аргумент формується рекурсивно нею ж, це рекурсія першого порядку.

Якщо функція викликає саму себе, а аргумент формується рекурсивно нею ж, де знову аргументом є рекурсивний виклик - це рекурсія другого порядку.



```
(define F (lambda (.....) (...(F (F.....) ...))))  
(define F (lambda (.....) (...(F (F(F.....) ...))))))
```



Прослідкувати рекурсію подумки не так і просто.

Тому варто дотримуватись певних рекомендацій стосовно техніки програмування рекурсивних обчислень.



# Рекомендація 1

Чітке планування термінальних гілок на основі розгляду найпростіших випадків.

Якщо гілок завершення декілька, розміщуйте спочатку найпростішу.

# Приклади термінальних гілок

- Ціль знайдено і потрібно повернути відповідь.
- Ціль не знайдено і немає більше елементів.

# Рекомендація 2.

При програмуванні рекурсивних гілок, враховуйте:

- Як спростити аргумент, наближуючи його крок за кроком до кінцевого значення (наближення до термінальної гілки).
- Форму, яку називають рекурсивним відношенням, яка пов'язує правильне значення поточного виклику зі значенням рекурсивного виклику.