***Сравнение циклических конструкций Common Lisp***

Циклические конструкции:

* DO
* DO\*
* DOTIMES
* PROG + GO

Код для каждой конструкции:

**1. Цикл DO**

*(defun factorial-do (n)*

*(do ((result 1 (\* result i))*

*(i 1 (+ i 1)))*

*((> i n) result)))*

*(time (factorial-prog 100000))*

**2. Цикл DO\***

*(defun factorial-do\* (n)*

*(do\* ((result 1 (\* result i))*

*(i 1 (+ i 1)))*

*((> i n) result)))*

*(time (factorial-prog 100000))*

**3. Цикл DOTIMES**

*(defun factorial-dotimes (n)*

*(let ((result 1))*

*(dotimes (i n result)*

*(setq result (\* result (+ i 1))))))*

*(time (factorial-prog 100000))*

**4. Цикл PROG с GO**

*(defun factorial-prog (n)*

*(prog (result i)*

*(setq result 1)*

*(setq i 1)*

*loop*

*(cond ((> i n) (return result)))*

*(setq result (\* result i))*

*(setq i (+ i 1))*

*(go loop)))*

*(time (factorial-prog 100000))*

**Сравнение при n = 100000**

|  |  |
| --- | --- |
| *Конструкция* | *Скорость* |
| DO | 11,48 |
| DO\* | 10,73 |
| DOTIMES | 6,19 |
| PROG + GO | 16,50 |

**DO**

Плюсы:

- Классический Lisp-стиль

- Параллельное связывание переменных

- Работает во всех реализациях

Минусы:

- Многословный синтаксис

- Средняя производительность (11.48 сек)

**DO\***

Плюсы:

- Последовательное связывание переменных

- Удобен для зависимых вычислений

- Лучше DO по производительности (10.73 сек)

Минусы:

- Тот же многословный синтаксис

- Не всегда очевидна разница с DO

**DOTIMES**

Плюсы:

- Наилучшая производительность (4.88 сек)

- Чистый и читаемый синтаксис

- Специализирован для численных циклов

Минусы:

- Ограничен только численными циклами

- Менее гибкий чем DO

**PROG + GO**

Плюсы:

- Полная свобода управления потоком

- Можно создавать сложные схемы переходов

Минусы:

- Наихудшая производительность (16.50 сек)

- Устаревший стиль программирования

- Низкая читаемость кода

- Сложность отладки

**Итог:**

DOTIMES - оптимальный выбор для численных циклов