1. Klokun

1.1. Опишіть вашу предметну область словами

Предметна область «Стек задач».

Об’єкти:

1. Стек — структура «останній прийшов — перший вийшов» (LIFO, last in — first out). Містить ім’я та перелік поточних задач.
2. Задача — опис можливой задачі. Містить ID і параметри задачі: назву і тривалість у секундах.

Необхідні функції:

1. Перевірити, чи є задача в стеку.
2. Додати задачу на верх стека (push).
3. Видалити задачу з верху стека (pop).

Запити:

1. Знайти завдання зі списку, що виконуються довше заданого часу.

1.2. Запропонуйте подання [частини] даних Вашої задачі з використанням списків Прологу

Подамо відомості про задачі так, щоб їх параметри були списком:

```
% задача (ID, ім'я, тривалість виконання (с)).
task(0, [send_email, 10]).
task(1, [cleanup, 55]).
task(2, [backup_db, 100]).
```

Для подання стеку використаємо список, щоб представити перелік задач.

```
% стек(name, [tasks]).
% стек(ім'я, список завдань у черзі за ID).
stack(s0, [0, 1, 2, 3]).
stack(s1, [1, 0, 3]).
stack(s2, [1, 2, 3]).
```

1.3. Запропонуйте предикати для розв'язання одного з запитів Вашої задачі з використанням списків Прологу.

Реалізуємо запит для пошуку у стеку завдань, що виконуються довше заданого часу:

```
% Рекурсивна перевірка, чи є завдання в стеці
% task_in_stack(завдання, стек)
task_in_stack(Task, [Task | _]).
task_in_stack(Task, [_ | T]) :-
    task_in_stack(Task, T).

% Пошук завдань, що виконуються довше заданого часу
% DurationThreshold
tasks_take_longer_than(StackName, Task, DurationThreshold) :-
    % Отримати зміст завдань в стеку
    stack(StackName, StackContents),
    % Знайти завдання у змісті стеку
    task_in_stack(Task, StackContents),
    % Знайти тривалість завдання
    task(Task, [_, Duration]),
    % Істина, якщо тривалість більша за поріг.
    Duration > DurationThreshold.
```

1.4. Запропонуйте подання [частини] даних Вашої задачі з використанням динамічних баз даних Прологу.

Робота з базами даних виконується на SWI-Prolog. Для цього оголосимо динамічні предикати — ті, що працюють з динамічними базами даних:

```
:- dynamic
    stack/2,
    task/2.
```

Знання про завдання і стеки оголошуються так само, але тепер вони зберігаються в неіменованій динамічній базі даних:

```
task(0, [send_email, 10]).
task(1, [cleanup, 55]).
task(2, [backup_db, 100]).
```

```
task(3, [run_checks, 50]).

stack(s0, [0, 1, 2, 3]).
stack(s1, [1, 0, 3]).
stack(s2, [1, 2, 3]).
```

1.5. Запропонуйте предикат(и) для розв'язання одного з запитів Вашої задачі з використанням динамічних баз даних Прологу

Оголосимо предикат, який витягує зі стеку верхній елемент (pop) і змінює даний стек у динамічній базі даних:

```
% Витягнути верхній елемент і повернути новий стек
% `NewStack` та витягнутий елемент `PoppedItem`
% Прототип:
% pop(Stack, NewStack (еквів. StackTail), PoppedItem (еквів. StackHead)).
pop([StackHead | StackTail], StackTail, StackHead).

% Витягнути верхній елемент стеку і записати
% отриманий стек в динамічну базу даних
pop_db(StackName) :-
    % Знайти стек і його зміст за ім'ям
    stack(StackName, Contents),
    % Витягнути верхній елемент
    pop(Contents, NewContents, _),
    % Видалити знання про минулий стек з бази даних
    retract(stack(StackName, _)),
    % І замінити їх новими
    assertz(stack(StackName, NewContents)).
```

1.6. Запропонуйте опис [частини] даних Вашої задачі з використанням засобів мови Ліспу.

Будемо представляти відомості про завдання (task). Мовою Пролог вони описані так:

```
task(0, [send_email, 10]).
task(1, [cleanup, 55]).
```

```
task(2, [backup_db, 100]).
task(3, [run_checks, 50]).
```

Тоді мовою Lisp:

```
(' (0 (send_email 10)))
(' (1 (cleanup 55)))
(' (2 (backup_db 100)))
(' (3 (run_checks 50)))
```