PROJECT REPORT

PROJECT TITLE – TO-DO LIST CONSOLE PLANNAR

NAME- ISHITA SAHA

REGISTRATION NUMBER – 25BCY10122

PROGRAM – B.TECH

COURSE CODE – CSE2001

COURSE NAME – CSE(CYBER SECURITY AND DIGITAL FORENSICS)

FACULTY – DR VISHNU VARTHANAN

ACADEMIC YEAR – 2025 – 29

INTRODUCTION

This project delivers a Dynamic To-Do List Console Agent, an user friendly interactive, menu-driven knowledge-based system built through Prolog logic programming language. It addresses the real world need for an efficient and reliable task manager.

The system defines a dynamic task/3 predicate to represent eacg to-do item with their unique ID in sequential, non-repeating way. The system supports user input for eg: adding new tasks. The system performs other querying and organising tasks to view tasks, marking them as complete or to completely deleting the tasks.

This system serves as a powerful practical demonstration of logic programming applied to solve everyday problems.

PROBLEM STATEMENT

Design and implement an intelligent system that allows a user to create, view, update and delete tasks through a simple console based logic. The system should assign each task a unique ID and track its status.

Constraints: Unique ID enforcement, Status of task, Input from the user,

OBJECTIVES

1.To develop a console-based To-Do List plannar that allows users to add,view,update and delete tasks in an organised way.

2.To implement dynamic task storage using Prolog predicates.

3.To provide a simple, menu-driven interface that guides users through all available operations.

4.To determine the use of Prolog concepts such as dynamic facts, recursion and input/output in a practical, real world style application.

NON – FUNCTIONAL REQUIREMENTS

- **Performance:** The system shall respond promptly to user inputs.

- **Usability:** The system shall have a simple, clear and user-friendly text-based menu interface that guides the user.

- **Reliability:** The system shall handle invalid inputs gracefully, displaying appropriate error messages.

- **Maintainability:** The code shall be modular with clear predicate definitions.

PROLOG CODE

```
% Prolog To-Do List Console Application

:- dynamic task/3.



load_tasks :-
    retractall(task(_, _, _)),
    write('Starting a new session. Dynamic memory cleared.'), nl.



save_tasks :-
    write('NOTE: File saving is disabled in this environment (SWISH sandbox).'), nl,
    write('Your current tasks will be lost when the session ends.'), nl.



next_id(ID) :-
    % Find all existing IDs
    findall(TID, task(TID, _, _), IDs),
    (   IDs = []
   ->  % If no tasks exist, start at 1
```

```
        ID = 1
    ;   % Otherwise, the new ID is the maximum existing ID + 1
        max_list(IDs, MaxID),
        ID is MaxID + 1
    ).

% add_task/0: Prompts for a new task description and adds it to the database.
add_task :-
    write('Enter task description: '),
    % NOTE: SWISH requires explicit input termination (like a period) for 'read_line_to_string'
    read_line_to_string(user_input, Description), % Read input as a string
    next_id(ID),
    asserta(task(ID, Description, pending)), % Assert the new fact (task)
    format('Task ~w added: "~w"', [ID, Description]), nl.

% view_tasks/0: Displays all tasks, categorized by status.
view_tasks :-
    nl, write('--- CURRENT TO-DO LIST---'), nl,

    % Find all pending tasks and sort by ID
    findall(task(ID, Desc, pending), task(ID, Desc, pending), Pending),
    write('PENDING Tasks:'), nl,
    (   Pending = []
    ->  write('    No pending tasks!'), nl
    ;   list_tasks(Pending)
    ),

    % Find all complete tasks
    findall(task(ID, Desc, complete), task(ID, Desc, complete), Complete),
    nl, write('COMPLETED Tasks:'), nl,
    (   Complete = []
    ->  write('    No completed tasks.'), nl
    ;   list_tasks(Complete)
    ),
    nl.

% list_tasks(+List): Helper to display a list of task facts.
list_tasks([]) :- !.
list_tasks([task(ID, Desc, Status)|Rest]) :-
    % Format output based on status
    ( Status = pending-> Prefix = '[ ]' ; Prefix = '[X]' ),
    format('    ~w [ID: ~w] ~w', [Prefix, ID, Desc]), nl,
    list_tasks(Rest).

% mark_complete/0: Changes a task's status from pending to complete.
mark_complete :-
    write('Enter ID of task to mark as COMPLETE (e.g., 1.): '),
    read(ID),
    (   task(ID, Desc, pending) % Check if the task exists and is pending
    ->  retract(task(ID, Desc, pending)), % Remove the old fact
        asserta(task(ID, Desc, complete)), % Assert the new (updated) fact
        format('Task ~w ("~w") marked as COMPLETE.', [ID, Desc]), nl
    ;   write('Error: Task ID not found or already complete.'), nl
    ).

% delete_task/0: Removes a task completely from the database.
delete_task :-
    write('Enter ID of task to DELETE (e.g., 1.): '),
    read(ID),
    (   task(ID, Desc, _) % Check if the task exists (regardless of status)
    ->  retract(task(ID, Desc, _)), % Remove the fact
        format('Task ~w ("~w") permanently DELETED.', [ID, Desc]), nl
    ;   write('Error: Task ID not found.'), nl
    ).


%------------------------------------------------------------
% 4. MENU AND EXECUTION LOOP
%------------------------------------------------------------

% menu/0: Displays the main menu options.
menu :-
    nl,
    write('--- Task Agent Menu---'), nl,
    write('1. View all tasks'), nl,
    write('2. Add new task'), nl,
    write('3. Mark task as complete'), nl,
    write('4. Delete task'), nl,
    write('5. Save & Exit (NOTE: Tasks will be lost)'), nl,
    write('6. Exit without Saving (Tasks will be lost)'), nl,
    write('Enter option (1-6): '),
    read(Option),
    process_option(Option).

% process_option(+Option): Executes the chosen action.
process_option(1) :- view_tasks, menu.
process_option(2) :- add_task, menu.
process_option(3) :- mark_complete, menu.

process_option(4) :- delete_task, menu.

% Option 5 and 6 now behave identically by exiting the loop.

process_option(5) :- save_tasks, write('Goodbye!'), nl.

process_option(6) :- write('Exiting.'), nl.

process_option(_) :-

    write('Invalid option. Please enter a number from 1 to 6.'), nl,

    menu.


% start/0: Entry point for the application.

start :-

    load_tasks, % Clear previous state

    menu.       % Start the main loop
```

OUTPUT

```
--- Task Agent Menu---
1. View all tasks
2. Add new task
3. Mark task as complete
4. Delete task
5. Save & Exit (NOTE: Tasks will be lost)
```

```
6. Exit without Saving (Tasks will be lost)
Enter option (1-6):

1


--- CURRENT TO-DO LIST---
PENDING Tasks:
No pending tasks!

COMPLETED Tasks:
No completed tasks.



--- Task Agent Menu---
1. View all tasks
2. Add new task
3. Mark task as complete
4. Delete task
5. Save & Exit (NOTE: Tasks will be lost)
6. Exit without Saving (Tasks will be lost)
Enter option (1-6):

1


--- CURRENT TO-DO LIST---
PENDING Tasks:
No pending tasks!
```

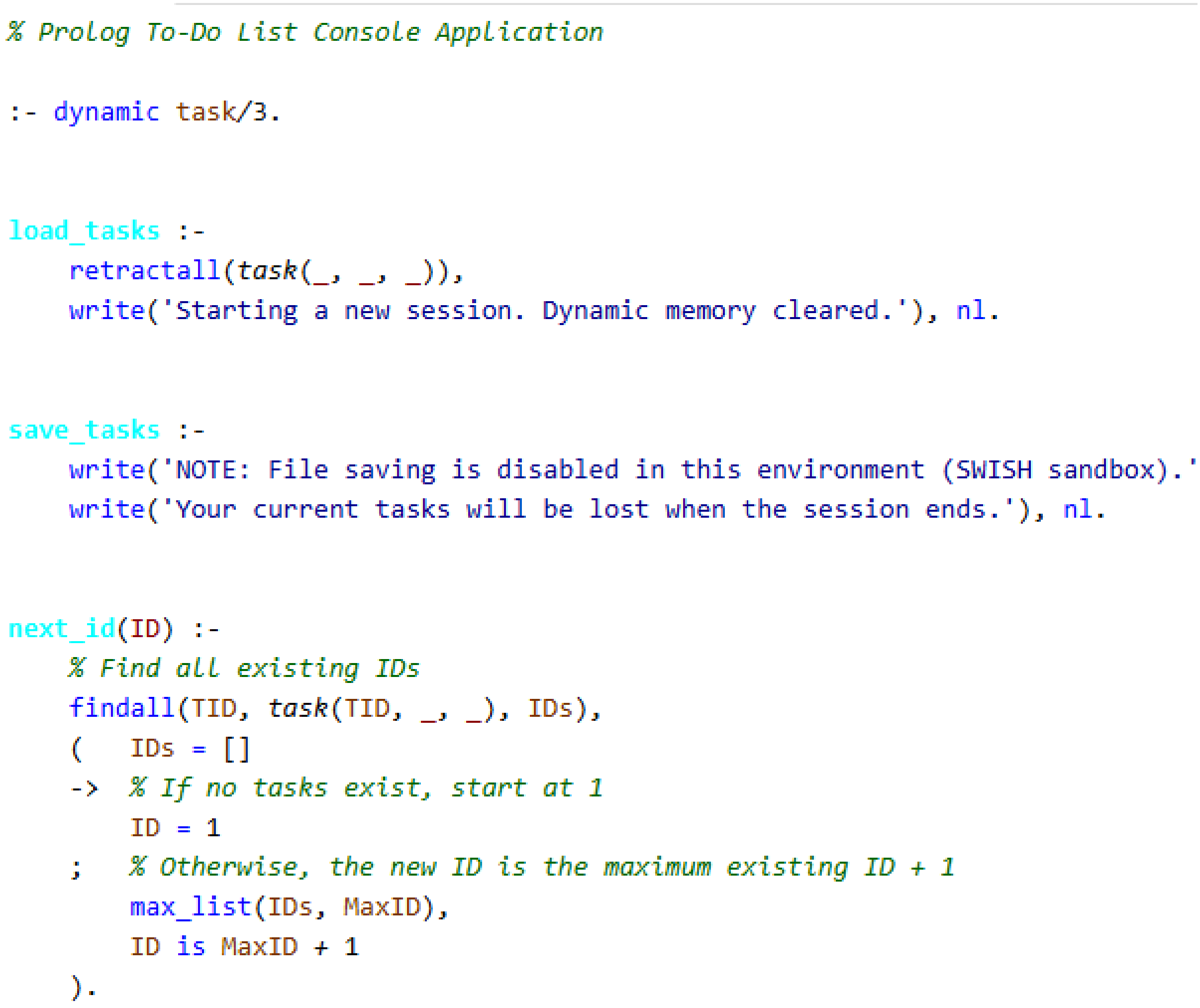
```
COMPLETED Tasks:
No completed tasks.


--- Task Agent Menu---
1. View all tasks
2. Add new task
3. Mark task as complete
4. Delete task
5. Save & Exit (NOTE: Tasks will be lost)
6. Exit without Saving (Tasks will be lost)
Enter option (1-6): 
start


Starting a new session. Dynamic memory cleared.

--- Task Agent Menu---
1. View all tasks
2. Add new task
3. Mark task as complete
4. Delete task
5. Save & Exit (NOTE: Tasks will be lost)
6. Exit without Saving (Tasks will be lost)
Enter option (1-6):
2
Enter task description:
```

```
read book
Task 1 added: "read book"

--- Task Agent Menu---
1. View all tasks
2. Add new task
3. Mark task as complete
4. Delete task
5. Save & Exit (NOTE: Tasks will be lost)
6. Exit without Saving (Tasks will be lost)
Enter option (1-6):
1


--- CURRENT TO-DO LIST---
PENDING Tasks:
    [ ] [ID: 1] read book

COMPLETED Tasks:
No completed tasks.



--- Task Agent Menu---
1. View all tasks
2. Add new task
3. Mark task as complete
4. Delete task
```

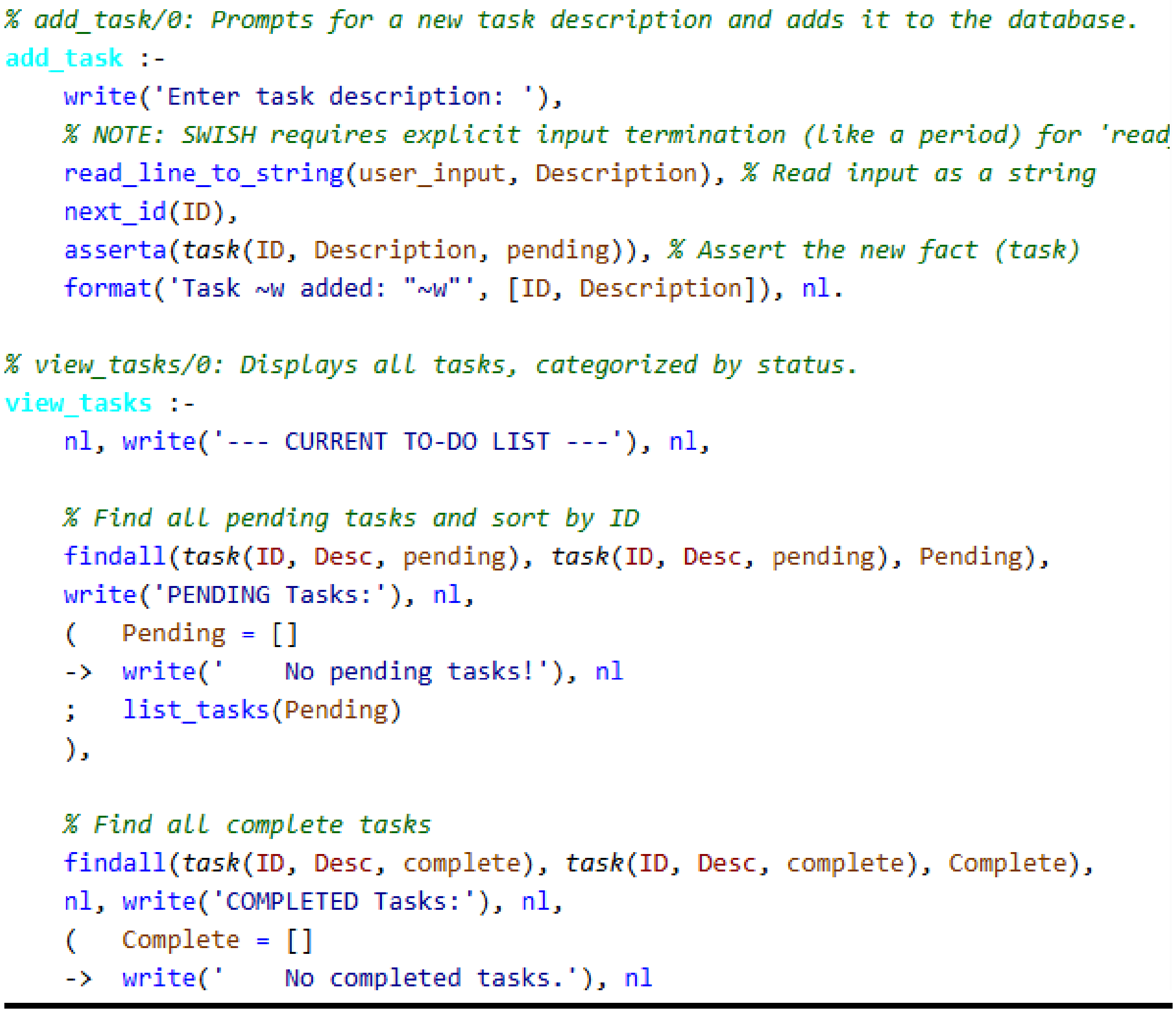
```
5. Save & Exit (NOTE: Tasks will be lost)
6. Exit without Saving (Tasks will be lost)
Enter option (1-6):

3

Enter ID of task to mark as COMPLETE (e.g., 1.):

1

Task 1 ("read book") marked as COMPLETE.


--- Task Agent Menu---
1. View all tasks
2. Add new task
3. Mark task as complete
4. Delete task
5. Save & Exit (NOTE: Tasks will be lost)
6. Exit without Saving (Tasks will be lost)
```

SNAPSHOT

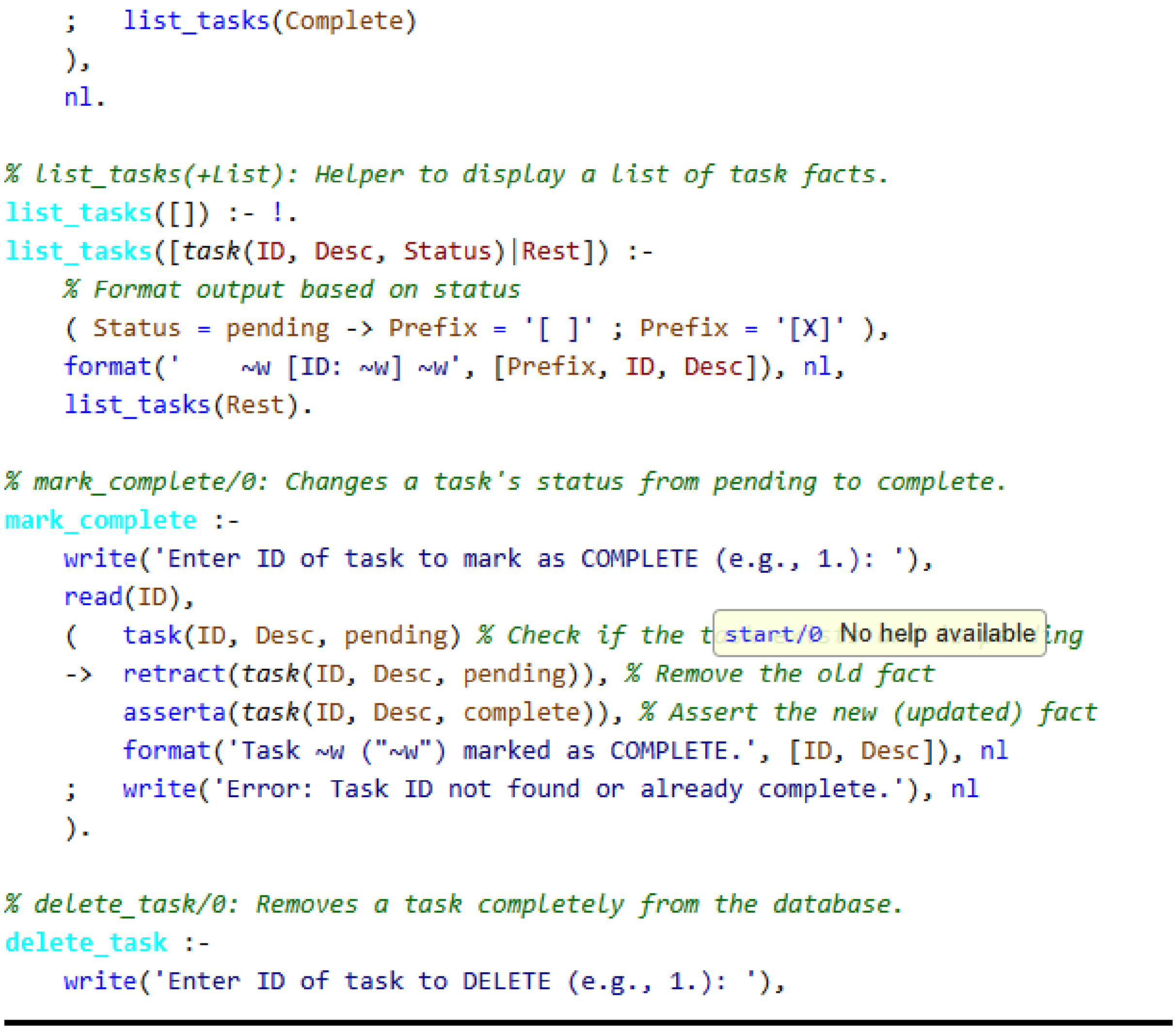
CODE:



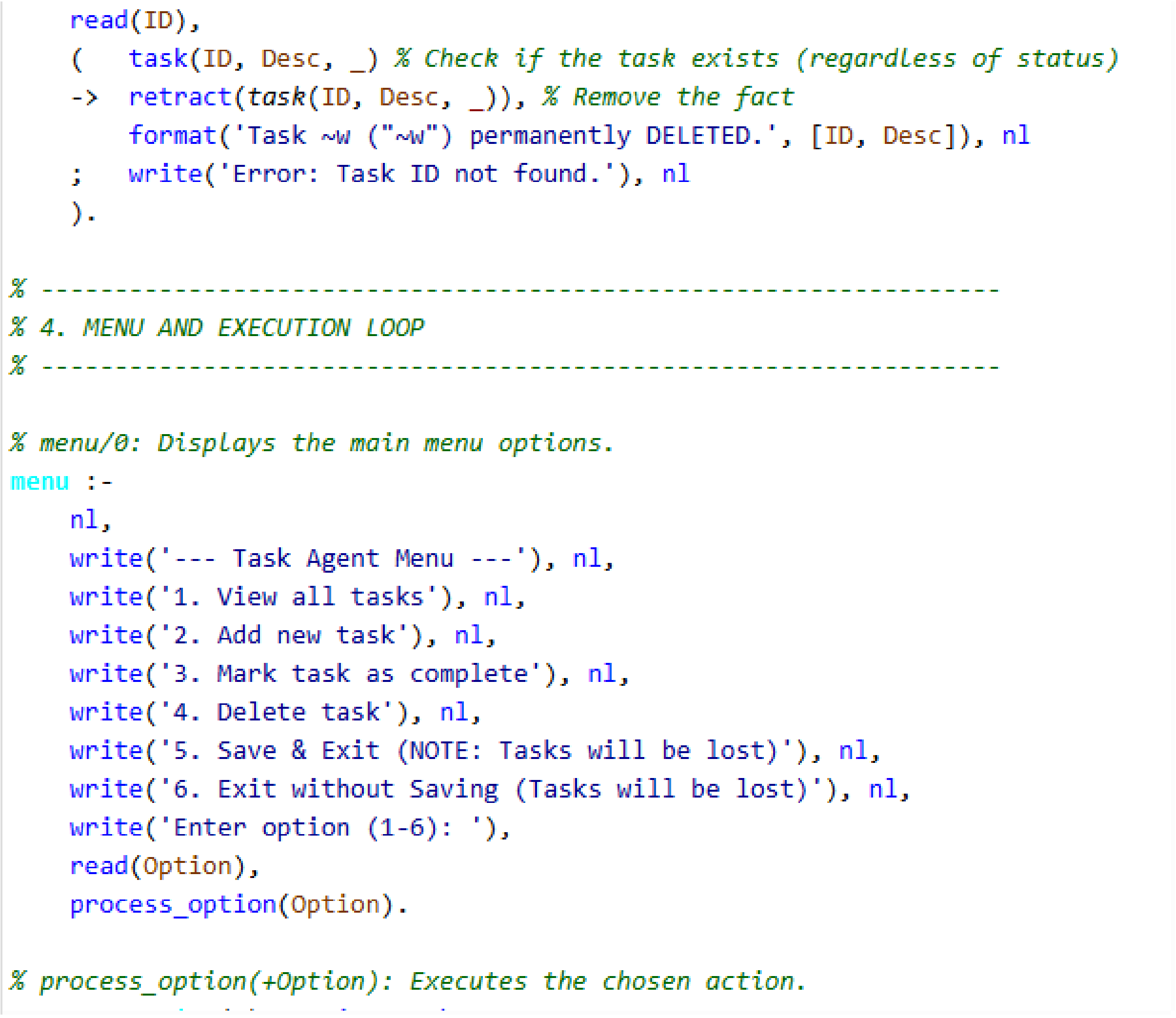
```prolog
% Prolog To-Do List Console Application

:- dynamic task/3.


load_tasks :-
    retractall(task(_, _, _)),
    write('Starting a new session. Dynamic memory cleared.'), nl.


save_tasks :-
    write('NOTE: File saving is disabled in this environment (SWISH sandbox).'
    write('Your current tasks will be lost when the session ends.'), nl.


next_id(ID) :-
    % Find all existing IDs
    findall(TID, task(TID, _, _), IDs),
    (   IDs = []
    ->  % If no tasks exist, start at 1
        ID = 1
    ;   % Otherwise, the new ID is the maximum existing ID + 1
        max_list(IDs, MaxID),
        ID is MaxID + 1
    ).
```



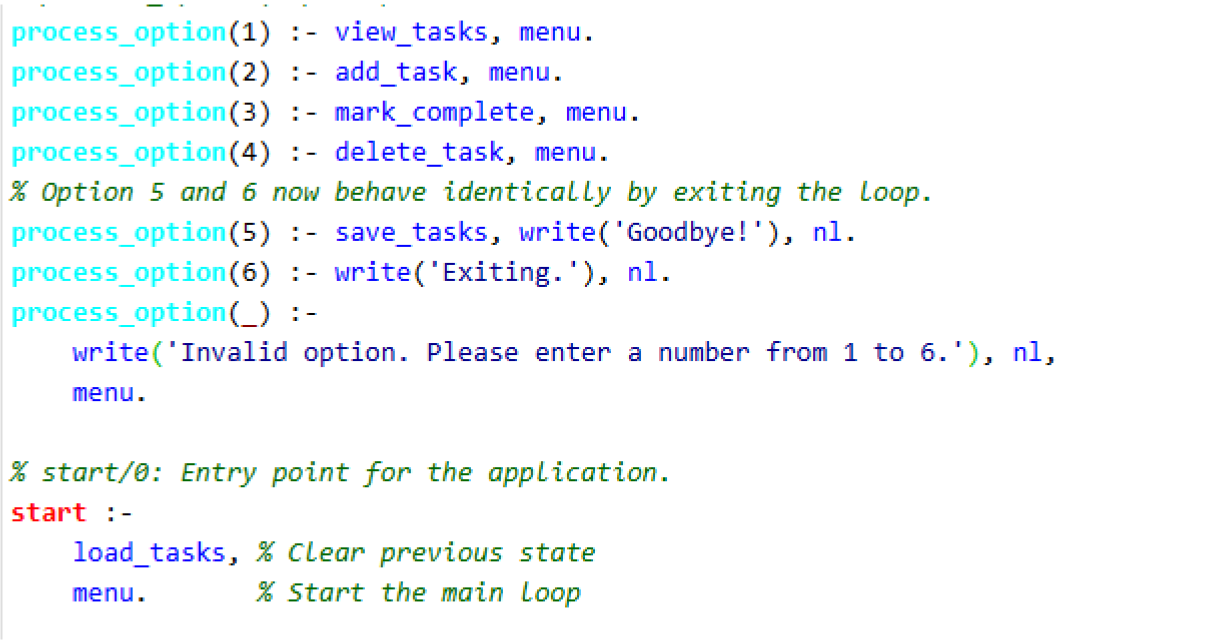
```prolog
% add_task/0: Prompts for a new task description and adds it to the database.
add_task :-
    write('Enter task description: '),
    % NOTE: SWISH requires explicit input termination (like a period) for 'read
    read_line_to_string(user_input, Description), % Read input as a string
    next_id(ID),
    asserta(task(ID, Description, pending)), % Assert the new fact (task)
    format('Task ~w added: "~w"', [ID, Description]), nl.

% view_tasks/0: Displays all tasks, categorized by status.
view_tasks :-
    nl, write('--- CURRENT TO-DO LIST ---'), nl,

    % Find all pending tasks and sort by ID
    findall(task(ID, Desc, pending), task(ID, Desc, pending), Pending),
    write('PENDING Tasks:'), nl,
    (   Pending = []
    ->  write('    No pending tasks!'), nl
    ;   list_tasks(Pending)
    ),

    % Find all complete tasks
    findall(task(ID, Desc, complete), task(ID, Desc, complete), Complete),
    nl, write('COMPLETED Tasks:'), nl,
    (   Complete = []
    ->  write('    No completed tasks.'), nl
```



```
    ;   list_tasks(Complete)
    ),
    nl.

% list_tasks(+List): Helper to display a list of task facts.
list_tasks([]) :- !.
list_tasks([task(ID, Desc, Status)|Rest]) :-
    % Format output based on status
    ( Status = pending -> Prefix = '[ ]' ; Prefix = '[X]' ),
    format('    ~w [ID: ~w] ~w', [Prefix, ID, Desc]), nl,
    list_tasks(Rest).

% mark_complete/0: Changes a task's status from pending to complete.
mark_complete :-
    write('Enter ID of task to mark as COMPLETE (e.g., 1.): '),
    read(ID),
    (   task(ID, Desc, pending) % Check if the t start/0 No help available ing
    ->  retract(task(ID, Desc, pending)), % Remove the old fact
        asserta(task(ID, Desc, complete)), % Assert the new (updated) fact
        format('Task ~w ("~w") marked as COMPLETE.', [ID, Desc]), nl
    ;   write('Error: Task ID not found or already complete.'), nl
    ).

% delete_task/0: Removes a task completely from the database.
delete_task :-
    write('Enter ID of task to DELETE (e.g., 1.): '),
```

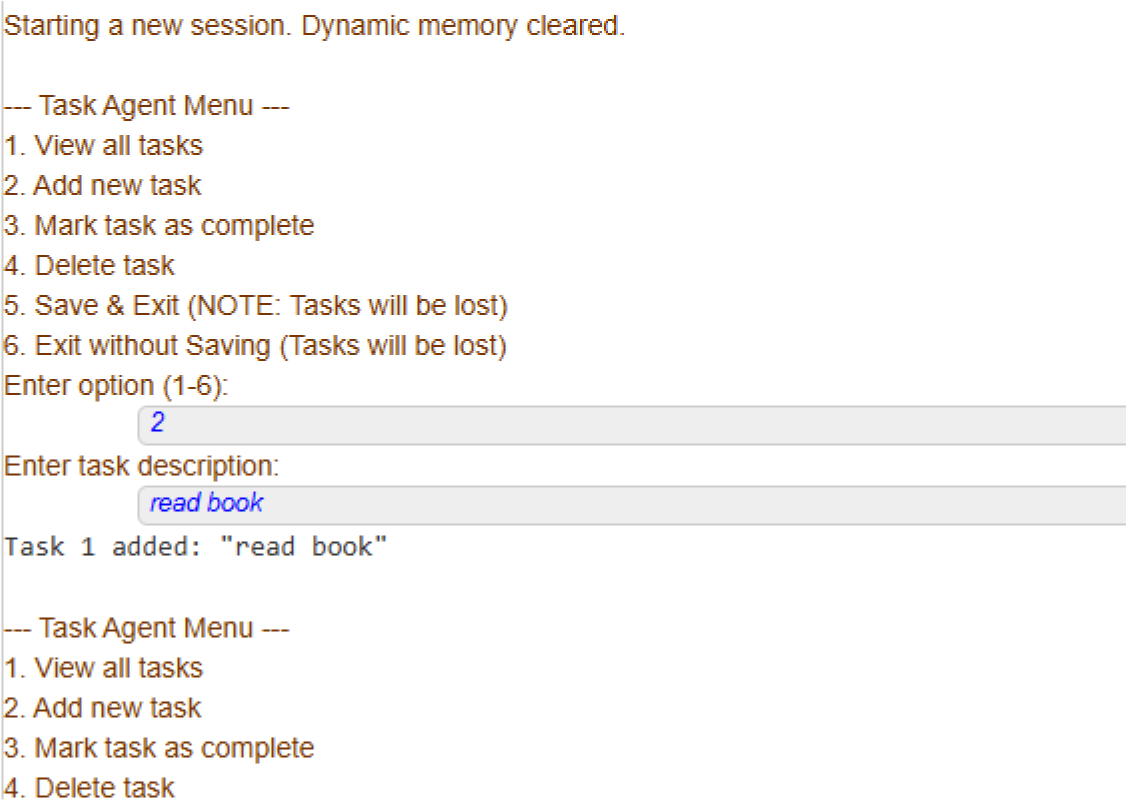


```
    read(ID),
    (   task(ID, Desc, _) % Check if the task exists (regardless of status)
    ->  retract(task(ID, Desc, _)), % Remove the fact
        format('Task ~w ("~w") permanently DELETED.', [ID, Desc]), nl
    ;   write('Error: Task ID not found.'), nl
    ).

% ----------------------------------------------------------------
% 4. MENU AND EXECUTION LOOP
% ----------------------------------------------------------------

% menu/0: Displays the main menu options.
menu :-
    nl,
    write('--- Task Agent Menu ---'), nl,
    write('1. View all tasks'), nl,
    write('2. Add new task'), nl,
    write('3. Mark task as complete'), nl,
    write('4. Delete task'), nl,
    write('5. Save & Exit (NOTE: Tasks will be lost)'), nl,
    write('6. Exit without Saving (Tasks will be lost)'), nl,
    write('Enter option (1-6): '),
    read(Option),
    process_option(Option).

% process_option(+Option): Executes the chosen action.
```

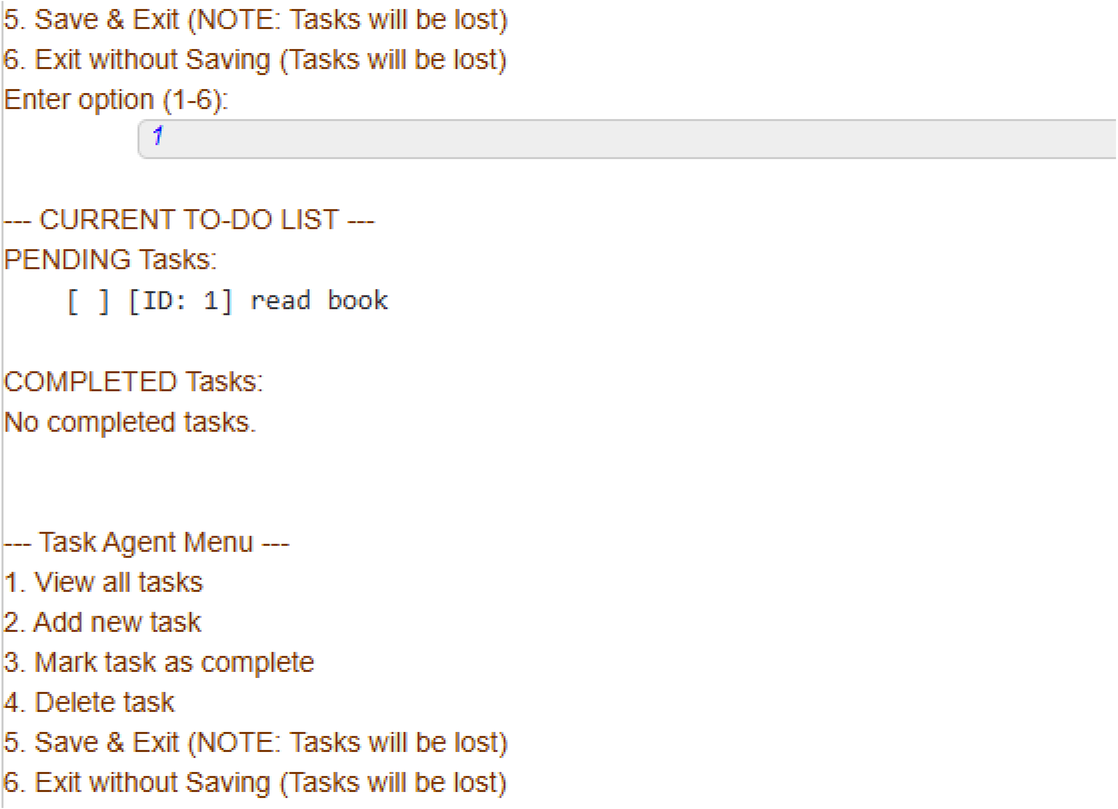


```
process_option(1) :- view_tasks, menu.
process_option(2) :- add_task, menu.
process_option(3) :- mark_complete, menu.
process_option(4) :- delete_task, menu.
% Option 5 and 6 now behave identically by exiting the loop.
process_option(5) :- save_tasks, write('Goodbye!'), nl.
process_option(6) :- write('Exiting.'), nl.
process_option(_) :-
    write('Invalid option. Please enter a number from 1 to 6.'), nl,
    menu.

% start/0: Entry point for the application.
start :-
    load_tasks, % Clear previous state
    menu.       % Start the main loop
```

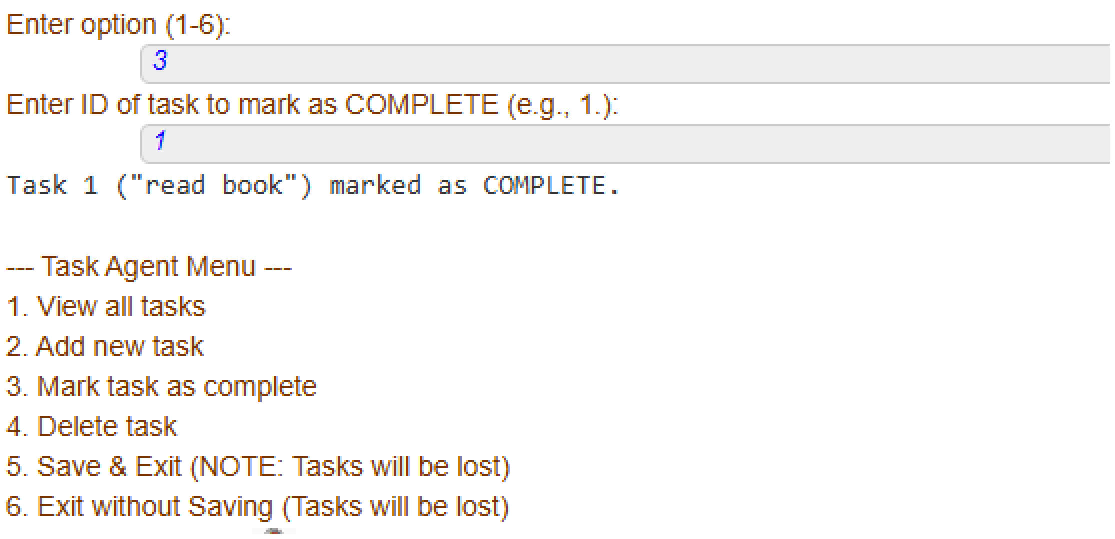
OUTPUT



```
Starting a new session. Dynamic memory cleared.

--- Task Agent Menu ---
1. View all tasks
2. Add new task
3. Mark task as complete
4. Delete task
5. Save & Exit (NOTE: Tasks will be lost)
6. Exit without Saving (Tasks will be lost)
Enter option (1-6):
          2
Enter task description:
          read book
Task 1 added: "read book"

--- Task Agent Menu ---
1. View all tasks
2. Add new task
3. Mark task as complete
4. Delete task
```



```
5. Save & Exit (NOTE: Tasks will be lost)
6. Exit without Saving (Tasks will be lost)
Enter option (1-6):
          1

--- CURRENT TO-DO LIST ---
PENDING Tasks:
    [ ] [ID: 1] read book

COMPLETED Tasks:
No completed tasks.


--- Task Agent Menu ---
1. View all tasks
2. Add new task
3. Mark task as complete
4. Delete task
5. Save & Exit (NOTE: Tasks will be lost)
6. Exit without Saving (Tasks will be lost)
```



```
Enter option (1-6):
3
Enter ID of task to mark as COMPLETE (e.g., 1.):
1
Task 1 ("read book") marked as COMPLETE.

--- Task Agent Menu ---
1. View all tasks
2. Add new task
3. Mark task as complete
4. Delete task
5. Save & Exit (NOTE: Tasks will be lost)
6. Exit without Saving (Tasks will be lost)
```

TESTING APPROACH

Unit tests:

- Test core predicates individually.
- Verify that each predicate handles both normal and edge cases(eg- adding the first task, marking a non-existing task).

Integration tests:

- Test sequences of operations to ensure correct transitions between states.

Validation tests:

- Confirm that all input handling properly validates user input and responds with appropriate prompts or error messages.
- Ensure the system runs smoothly in the intended environment.

CHALLENGES FACED

- Dynamic Memory Management
- Input Validation
- Session Persistence Limitations
- User Interface Design
- Unique ID Generation

LEARNING KEY AND TAKEAWAYS

- Gained practical experience using assert, retract, and dynamic predicates to manage runtime-modifiable facts.
- Developed skills in building user-friendly menu driven interfaces.
- Learned how to handle stateful data in a stateless language context.
- Understood the importance of validating user inputs and managing error scenarios gracefully to build reliable, user-friendly applications.

FUTURE ENHANCEMENTS

- Integrating the file-based or database storage to save tasks beyond the current session.

- Adding attributes for task priority levels and deadlines, enabling sorting and reminders to handle urgent tasks more effectively.

- Incorporating estimated time requirements per task and notify users about overdue or upcoming tasks.