Problem 2590: Design a Todo List

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Design a Todo List Where users can add

tasks

, mark them as

complete

, or get a list of pending tasks. Users can also add

tags

to tasks and can filter the tasks by certain tags.

Implement the

TodoList

class:

TodoList()

Initializes the object.

int addTask(int userId, String taskDescription, int dueDate, List<String> tags)

Adds a task for the user with the ID

userId

with a due date equal to

dueDate

and a list of tags attached to the task. The return value is the ID of the task. This ID starts at

1

and is

sequentially

increasing. That is, the first task's id should be

1

, the second task's id should be

2

, and so on.

List<String> getAllTasks(int userId)

Returns a list of all the tasks not marked as complete for the user with ID

userId

, ordered by the due date. You should return an empty list if the user has no uncompleted tasks.

List<String> getTasksForTag(int userId, String tag)

Returns a list of all the tasks that are not marked as complete for the user with the ID userId

and have

tag

as one of their tags, ordered by their due date. Return an empty list if no such task exists.

void completeTask(int userId, int taskId)

Marks the task with the ID

taskId

as completed only if the task exists and the user with the ID

userId

has this task, and it is uncompleted.

Example 1:

Input

["TodoList", "addTask", "addTask", "getAllTasks", "getAllTasks", "addTask", "getTasksForTag", "completeTask", "completeTask", "getTasksForTag", "getAllTasks"] [[], [1, "Task1", 50, []], [1, "Task2", 100, ["P1"]], [1], [5], [1, "Task3", 30, ["P1"]], [1, "P1"], [5, 1], [1, 2], [1, "P1"], [1]]

Output

[null, 1, 2, ["Task1", "Task2"], [], 3, ["Task3", "Task2"], null, null, ["Task3"], ["Task3", "Task1"]]

Explanation

TodoList todoList = new TodoList(); todoList.addTask(1, "Task1", 50, []); // return 1. This adds a new task for the user with id 1. todoList.addTask(1, "Task2", 100, ["P1"]); // return 2. This adds another task for the user with id 1. todoList.getAllTasks(1); // return ["Task1", "Task2"]. User 1 has two uncompleted tasks so far. todoList.getAllTasks(5); // return []. User 5 does not have any tasks so far. todoList.addTask(1, "Task3", 30, ["P1"]); // return 3. This adds another task for the user with id 1. todoList.getTasksForTag(1, "P1"); // return ["Task3", "Task2"]. This returns the uncompleted tasks that have the tag "P1" for the user with id 1. todoList.completeTask(5, 1); // This does nothing, since task 1 does not belong to user 5. todoList.completeTask(1, 2); // This marks task 2 as completed. todoList.getTasksForTag(1, "P1"); // return ["Task3"]. This returns the uncompleted tasks that have the tag "P1" for the user with id 1. // Notice that we did not include "Task2" because it is completed now. todoList.getAllTasks(1); // return ["Task3", "Task1"]. User 1 now has 2 uncompleted tasks.

Constraints:

1 <= userId, taskId, dueDate <= 100

0 <= tags.length <= 100

1 <= taskDescription.length <= 50

1 <= tags[i].length, tag.length <= 20

All

dueDate

values are unique.

All the strings consist of lowercase and uppercase English letters and digits.

At most

100

calls will be made for each method.

Code Snippets

C++:

```
class TodoList {
public:
```

```
TodoList() {

}

int addTask(int userId, string taskDescription, int dueDate, vector<string>
tags) {

}

vector<string> getAllTasks(int userId) {

}

vector<string> getTasksForTag(int userId, string tag) {

}

void completeTask(int userId, int taskId) {

}
};

/**
* Your TodoList object will be instantiated and called as such:
* TodoList* obj = new TodoList();
* int param_1 = obj->addTask(userId,taskDescription,dueDate,tags);
* vector<string> param_2 = obj->getAllTasks(userId);
* vector<string> param_3 = obj->getTasksForTag(userId,tag);
* obj->completeTask(userId,taskId);
*/
```

Java:

```
class TodoList {

public TodoList() {

}

public int addTask(int userId, String taskDescription, int dueDate,
List<String> tags) {
}

public List<String> getAllTasks(int userId) {

}

public List<String> getTasksForTag(int userId, String tag) {

}

public void completeTask(int userId, int taskId) {

}
}

/**
* Your TodoList object will be instantiated and called as such:
* TodoList obj = new TodoList();
* int param_1 = obj.addTask(userId,taskDescription,dueDate,tags);
* List<String> param_2 = obj.getAllTasks(userId);
* List<String> param_3 = obj.getTasksForTag(userId,tag);
* obj.completeTask(userId,taskId);
*/
```

Python3:

```
class TodoList:

def __init__(self):



def addTask(self, userId: int, taskDescription: str, dueDate: int, tags:
List[str]) -> int:



def getAllTasks(self, userId: int) -> List[str]:



def getTasksForTag(self, userId: int, tag: str) -> List[str]:



def completeTask(self, userId: int, taskId: int) -> None:
```

```
# Your TodoList object will be instantiated and called as such:
# obj = TodoList()
# param_1 = obj.addTask(userId,taskDescription,dueDate,tags)
# param_2 = obj.getAllTasks(userId)
# param_3 = obj.getTasksForTag(userId,tag)
# obj.completeTask(userId,taskId)
```

Python:

```
class TodoList(object):

def __init__(self):



def addTask(self, userId, taskDescription, dueDate, tags):
"""
:type userId: int
:type taskDescription: str
:type dueDate: int
:type tags: List[str]
:rtype: int
"""



def getAllTasks(self, userId):
"""
:type userId: int
:rtype: List[str]
"""



def getTasksForTag(self, userId, tag):
"""
:type userId: int
:type tag: str
:rtype: List[str]
"""
```

```
def completeTask(self, userId, taskId):
"""
:type userId: int
:type taskId: int
:rtype: None
"""



# Your TodoList object will be instantiated and called as such:
# obj = TodoList()
# param_1 = obj.addTask(userId,taskDescription,dueDate,tags)
# param_2 = obj.getAllTasks(userId)
# param_3 = obj.getTasksForTag(userId,tag)
# obj.completeTask(userId,taskId)
```

JavaScript:

```
var TodoList = function() {

};

/**
* @param {number} userId
* @param {string} taskDescription
* @param {number} dueDate
* @param {string[]} tags
* @return {number}
*/
TodoList.prototype.addTask = function(userId, taskDescription, dueDate, tags)
{

};

/**
* @param {number} userId
* @return {string[]}
*/
TodoList.prototype.getAllTasks = function(userId) {

};
```

```
/**
* @param {number} userId
* @param {string} tag
* @return {string[]}
*/
TodoList.prototype.getTasksForTag = function(userId, tag) {

};

/**
* @param {number} userId
* @param {number} taskId
* @return {void}
*/
TodoList.prototype.completeTask = function(userId, taskId) {

};

/**
* Your TodoList object will be instantiated and called as such:
* var obj = new TodoList()
* var param_1 = obj.addTask(userId,taskDescription,dueDate,tags)
* var param_2 = obj.getAllTasks(userId)
* var param_3 = obj.getTasksForTag(userId,tag)
* obj.completeTask(userId,taskId)
*/
```

TypeScript:

```
class TodoList {
constructor() {

}

addTask(userId: number, taskDescription: string, dueDate: number, tags:
string[]): number {

}

getAllTasks(userId: number): string[] {
}

getTasksForTag(userId: number, tag: string): string[] {

}

completeTask(userId: number, taskId: number): void {

}
}

/**
* Your TodoList object will be instantiated and called as such:
* var obj = new TodoList()
* var param_1 = obj.addTask(userId,taskDescription,dueDate,tags)
* var param_2 = obj.getAllTasks(userId)
* var param_3 = obj.getTasksForTag(userId,tag)
* obj.completeTask(userId,taskId)
*/
```

C#:

```
public class TodoList {

public TodoList() {

}

public int AddTask(int userId, string taskDescription, int dueDate,
IList<string> tags) {

}

public IList<string> GetAllTasks(int userId) {

}

public IList<string> GetTasksForTag(int userId, string tag) {

}

public void CompleteTask(int userId, int taskId) {
}
}

/**
* Your TodoList object will be instantiated and called as such:
* TodoList obj = new TodoList();
* int param_1 = obj.AddTask(userId,taskDescription,dueDate,tags);
* IList<string> param_2 = obj.GetAllTasks(userId);
* IList<string> param_3 = obj.GetTasksForTag(userId,tag);
* obj.CompleteTask(userId,taskId);
*/
```

C:

```
typedef struct {

} TodoList;


TodoList* todoListCreate() {

}

int todoListAddTask(TodoList* obj, int userId, char* taskDescription, int
dueDate, char** tags, int tagsSize) {

}

char** todoListGetAllTasks(TodoList* obj, int userId, int* retSize) {

}

char** todoListGetTasksForTag(TodoList* obj, int userId, char* tag, int*
retSize) {

}

void todoListCompleteTask(TodoList* obj, int userId, int taskId) {
}

void todoListFree(TodoList* obj) {

}

/**
* Your TodoList struct will be instantiated and called as such:
* TodoList* obj = todoListCreate();
* int param_1 = todoListAddTask(obj, userId, taskDescription, dueDate, tags,
tagsSize);

* char** param_2 = todoListGetAllTasks(obj, userId, retSize);

* char** param_3 = todoListGetTasksForTag(obj, userId, tag, retSize);

* todoListCompleteTask(obj, userId, taskId);

* todoListFree(obj);
*/
```

Go:

```
type TodoList struct {

}


func Constructor() TodoList {

}


func (this *TodoList) AddTask(userId int, taskDescription string, dueDate
int, tags []string) int {

}


func (this *TodoList) GetAllTasks(userId int) []string {
}


func (this *TodoList) GetTasksForTag(userId int, tag string) []string {

}


func (this *TodoList) CompleteTask(userId int, taskId int) {

}


/**
* Your TodoList object will be instantiated and called as such:
* obj := Constructor();
* param_1 := obj.AddTask(userId,taskDescription,dueDate,tags);
* param_2 := obj.GetAllTasks(userId);
* param_3 := obj.GetTasksForTag(userId,tag);
* obj.CompleteTask(userId,taskId);
*/
```

Kotlin:

```
class TodoList() {

fun addTask(userId: Int, taskDescription: String, dueDate: Int, tags:
List<String>): Int {

}

fun getAllTasks(userId: Int): List<String> {

}

fun getTasksForTag(userId: Int, tag: String): List<String> {

}

fun completeTask(userId: Int, taskId: Int) {

}
}

/**
* Your TodoList object will be instantiated and called as such:
* var obj = TodoList()
* var param_1 = obj.addTask(userId,taskDescription,dueDate,tags)
* var param_2 = obj.getAllTasks(userId)
* var param_3 = obj.getTasksForTag(userId,tag)
* obj.completeTask(userId,taskId)
*/
```

Swift:

```
class TodoList {

init() {

}

func addTask(_ userId: Int, _ taskDescription: String, _ dueDate: Int, _
tags: [String]) -> Int {

}

func getAllTasks(_ userId: Int) -> [String] {

}

func getTasksForTag(_ userId: Int, _ tag: String) -> [String] {

}

func completeTask(_ userId: Int, _ taskId: Int) {

}
}

/**
* Your TodoList object will be instantiated and called as such:
* let obj = TodoList()
* let ret_1: Int = obj.addTask(userId, taskDescription, dueDate, tags)
* let ret_2: [String] = obj.getAllTasks(userId)
* let ret_3: [String] = obj.getTasksForTag(userId, tag)
* obj.completeTask(userId, taskId)
*/
```

Rust:

```
struct TodoList {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl TodoList {

fn new() -> Self {

}

fn add_task(&self, user_id: i32, task_description: String, due_date: i32,
tags: Vec<String>) -> i32 {

}

fn get_all_tasks(&self, user_id: i32) -> Vec<String> {

}

fn get_tasks_for_tag(&self, user_id: i32, tag: String) -> Vec<String> {

}

fn complete_task(&self, user_id: i32, task_id: i32) {

}
}

/**
```

```
* Your TodoList object will be instantiated and called as such:
* let obj = TodoList::new();
* let ret_1: i32 = obj.add_task(userId, taskDescription, dueDate, tags);
* let ret_2: Vec<String> = obj.get_all_tasks(userId);
* let ret_3: Vec<String> = obj.get_tasks_for_tag(userId, tag);
* obj.complete_task(userId, taskId);
*/
```

Ruby:

```
class TodoList
def initialize()

end



=begin
:type user_id: Integer
:type task_description: String
:type due_date: Integer
:type tags: String[]
:rtype: Integer
=end
def add_task(user_id, task_description, due_date, tags)

end



=begin
:type user_id: Integer
:rtype: String[]
=end
def get_all_tasks(user_id)

end



=begin
:type user_id: Integer
:type tag: String
:rtype: String[]
=end
```

```
def get_tasks_for_tag(user_id, tag)

end



=begin
:type user_id: Integer
:type task_id: Integer
:rtype: Void
=end
def complete_task(user_id, task_id)

end



end

# Your TodoList object will be instantiated and called as such:
# obj = TodoList.new()
# param_1 = obj.add_task(user_id, task_description, due_date, tags)
# param_2 = obj.get_all_tasks(user_id)
# param_3 = obj.get_tasks_for_tag(user_id, tag)
# obj.complete_task(user_id, task_id)
```

PHP:

```
class TodoList {
/**
*/
function __construct() {

}

/**
* @param Integer $userId
* @param String $taskDescription
* @param Integer $dueDate
* @param String[] $tags
* @return Integer
*/
function addTask($userId, $taskDescription, $dueDate, $tags) {
}

/**
* @param Integer $userId
* @return String[]
*/
function getAllTasks($userId) {

}

/**
* @param Integer $userId
* @param String $tag
* @return String[]
*/
function getTasksForTag($userId, $tag) {

}

/**
* @param Integer $userId
* @param Integer $taskId
* @return NULL
*/
function completeTask($userId, $taskId) {

}
}

/**
* Your TodoList object will be instantiated and called as such:
* $obj = TodoList();
* $ret_1 = $obj->addTask($userId, $taskDescription, $dueDate, $tags);
* $ret_2 = $obj->getAllTasks($userId);
* $ret_3 = $obj->getTasksForTag($userId, $tag);
* $obj->completeTask($userId, $taskId);
*/
```

Dart:

```
class TodoList {
```

```
TodoList() {

}

int addTask(int userId, String taskDescription, int dueDate, List<String>
tags) {

}

List<String> getAllTasks(int userId) {

}

List<String> getTasksForTag(int userId, String tag) {

}

void completeTask(int userId, int taskId) {

}
}

/**
* Your TodoList object will be instantiated and called as such:
* TodoList obj = TodoList();
* int param1 = obj.addTask(userId,taskDescription,dueDate,tags);
* List<String> param2 = obj.getAllTasks(userId);
* List<String> param3 = obj.getTasksForTag(userId,tag);
* obj.completeTask(userId,taskId);
*/
```

Scala:

```
class TodoList() {

def addTask(userId: Int, taskDescription: String, dueDate: Int, tags:
List[String]): Int = {

}

def getAllTasks(userId: Int): List[String] = {
}

def getTasksForTag(userId: Int, tag: String): List[String] = {

}

def completeTask(userId: Int, taskId: Int): Unit = {

}

}

/**
* Your TodoList object will be instantiated and called as such:
* val obj = new TodoList()
* val param_1 = obj.addTask(userId,taskDescription,dueDate,tags)
* val param_2 = obj.getAllTasks(userId)
* val param_3 = obj.getTasksForTag(userId,tag)
* obj.completeTask(userId,taskId)
*/
```

Elixir:

```
defmodule TodoList do
@spec init_() :: any
def init_() do

end

@spec add_task(user_id :: integer, task_description :: String.t, due_date ::
integer, tags :: [String.t]) :: integer
def add_task(user_id, task_description, due_date, tags) do

end

@spec get_all_tasks(user_id :: integer) :: [String.t]
def get_all_tasks(user_id) do

end

@spec get_tasks_for_tag(user_id :: integer, tag :: String.t) :: [String.t]
def get_tasks_for_tag(user_id, tag) do
```

```
end

@spec complete_task(user_id :: integer, task_id :: integer) :: any
def complete_task(user_id, task_id) do

end
end

# Your functions will be called as such:
# TodoList.init_()
# param_1 = TodoList.add_task(user_id, task_description, due_date, tags)
# param_2 = TodoList.get_all_tasks(user_id)
# param_3 = TodoList.get_tasks_for_tag(user_id, tag)
# TodoList.complete_task(user_id, task_id)

# TodoList.init_ will be called before every test case, in which you can do
some necessary initializations.
```

Erlang:

```
-spec todo_list_init_() -> any().
todo_list_init_() ->
.

-spec todo_list_add_task(UserId :: integer(), TaskDescription ::
unicode:unicode_binary(), DueDate :: integer(), Tags ::
[unicode:unicode_binary()]) -> integer().
todo_list_add_task(UserId, TaskDescription, DueDate, Tags) ->
.

-spec todo_list_get_all_tasks(UserId :: integer()) ->
[unicode:unicode_binary()].
todo_list_get_all_tasks(UserId) ->
.

-spec todo_list_get_tasks_for_tag(UserId :: integer(), Tag ::
unicode:unicode_binary()) -> [unicode:unicode_binary()].
todo_list_get_tasks_for_tag(UserId, Tag) ->
.

-spec todo_list_complete_task(UserId :: integer(), TaskId :: integer()) ->
```

```
any().
todo_list_complete_task(UserId, TaskId) ->
.



%% Your functions will be called as such:
%% todo_list_init_(),
%% Param_1 = todo_list_add_task(UserId, TaskDescription, DueDate, Tags),
%% Param_2 = todo_list_get_all_tasks(UserId),
%% Param_3 = todo_list_get_tasks_for_tag(UserId, Tag),
%% todo_list_complete_task(UserId, TaskId),

%% todo_list_init_ will be called before every test case, in which you can do
some necessary initializations.
```

Racket:

```
(define todo-list%
(class object%
(super-new)

(init-field)

; add-task : exact-integer? string? exact-integer? (listof string?) ->
exact-integer?
(define/public (add-task user-id task-description due-date tags)
)
; get-all-tasks : exact-integer? -> (listof string?)
(define/public (get-all-tasks user-id)
)
; get-tasks-for-tag : exact-integer? string? -> (listof string?)
(define/public (get-tasks-for-tag user-id tag)
)
; complete-task : exact-integer? exact-integer? -> void?
(define/public (complete-task user-id task-id)
)))

;; Your todo-list% object will be instantiated and called as such:
;; (define obj (new todo-list%))
;; (define param_1 (send obj add-task user-id task-description due-date
tags))
;; (define param_2 (send obj get-all-tasks user-id))
;; (define param_3 (send obj get-tasks-for-tag user-id tag))
;; (send obj complete-task user-id task-id)
```

Solutions

C++ Solution:

```
/*
* Problem: Design a Todo List
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class TodoList {
public:
TodoList() {

}

int addTask(int userId, string taskDescription, int dueDate, vector<string>
tags) {

}

vector<string> getAllTasks(int userId) {

}

vector<string> getTasksForTag(int userId, string tag) {

}

void completeTask(int userId, int taskId) {

}
};
```

```
/**
* Your TodoList object will be instantiated and called as such:
* TodoList* obj = new TodoList();
* int param_1 = obj->addTask(userId,taskDescription,dueDate,tags);
* vector<string> param_2 = obj->getAllTasks(userId);
* vector<string> param_3 = obj->getTasksForTag(userId,tag);
* obj->completeTask(userId,taskId);
*/
```

Java Solution:

```
/**
* Problem: Design a Todo List
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class TodoList {

public TodoList() {

}

public int addTask(int userId, String taskDescription, int dueDate,
List<String> tags) {

}

public List<String> getAllTasks(int userId) {

}

public List<String> getTasksForTag(int userId, String tag) {

}

public void completeTask(int userId, int taskId) {
}
}

/**
* Your TodoList object will be instantiated and called as such:
* TodoList obj = new TodoList();
* int param_1 = obj.addTask(userId,taskDescription,dueDate,tags);
* List<String> param_2 = obj.getAllTasks(userId);
* List<String> param_3 = obj.getTasksForTag(userId,tag);
* obj.completeTask(userId,taskId);
*/
```

Python3 Solution:

```
"""
Problem: Design a Todo List
Difficulty: Medium
Tags: array, string, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class TodoList:

def __init__(self):



def addTask(self, userId: int, taskDescription: str, dueDate: int, tags:
List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class TodoList(object):

def __init__(self):
```

```
def addTask(self, userId, taskDescription, dueDate, tags):
"""
:type userId: int
:type taskDescription: str
:type dueDate: int
:type tags: List[str]
:rtype: int
"""


def getAllTasks(self, userId):
"""
:type userId: int
:rtype: List[str]
"""


def getTasksForTag(self, userId, tag):
"""
:type userId: int
:type tag: str
:rtype: List[str]
"""


def completeTask(self, userId, taskId):
"""
:type userId: int
:type taskId: int
:rtype: None
"""



# Your TodoList object will be instantiated and called as such:
# obj = TodoList()
# param_1 = obj.addTask(userId,taskDescription,dueDate,tags)
# param_2 = obj.getAllTasks(userId)
# param_3 = obj.getTasksForTag(userId,tag)
# obj.completeTask(userId,taskId)
```

JavaScript Solution:

```
/**
* Problem: Design a Todo List
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/


var TodoList = function() {

};

/**
* @param {number} userId
* @param {string} taskDescription
* @param {number} dueDate
* @param {string[]} tags
* @return {number}
*/
TodoList.prototype.addTask = function(userId, taskDescription, dueDate, tags)
{

};

/**
* @param {number} userId
* @return {string[]}
*/
TodoList.prototype.getAllTasks = function(userId) {

};

/**
* @param {number} userId
* @param {string} tag
* @return {string[]}
*/
```

```
TodoList.prototype.getTasksForTag = function(userId, tag) {

};

/**
* @param {number} userId
* @param {number} taskId
* @return {void}
*/
TodoList.prototype.completeTask = function(userId, taskId) {

};

/**
* Your TodoList object will be instantiated and called as such:
* var obj = new TodoList()
* var param_1 = obj.addTask(userId,taskDescription,dueDate,tags)
* var param_2 = obj.getAllTasks(userId)
* var param_3 = obj.getTasksForTag(userId,tag)
* obj.completeTask(userId,taskId)
*/
```

TypeScript Solution:

```
/**
* Problem: Design a Todo List
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class TodoList {
constructor() {

}

addTask(userId: number, taskDescription: string, dueDate: number, tags:
string[]): number {
}

getAllTasks(userId: number): string[] {

}

getTasksForTag(userId: number, tag: string): string[] {

}

completeTask(userId: number, taskId: number): void {

}
}

/**
* Your TodoList object will be instantiated and called as such:
* var obj = new TodoList()
* var param_1 = obj.addTask(userId,taskDescription,dueDate,tags)
* var param_2 = obj.getAllTasks(userId)
* var param_3 = obj.getTasksForTag(userId,tag)
* obj.completeTask(userId,taskId)
*/
```

C# Solution:

```
/*
* Problem: Design a Todo List
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class TodoList {

public TodoList() {
}

public int AddTask(int userId, string taskDescription, int dueDate,
IList<string> tags) {

}

public IList<string> GetAllTasks(int userId) {

}

public IList<string> GetTasksForTag(int userId, string tag) {

}

public void CompleteTask(int userId, int taskId) {

}
}

/**
* Your TodoList object will be instantiated and called as such:
* TodoList obj = new TodoList();
* int param_1 = obj.AddTask(userId,taskDescription,dueDate,tags);
* IList<string> param_2 = obj.GetAllTasks(userId);
* IList<string> param_3 = obj.GetTasksForTag(userId,tag);
* obj.CompleteTask(userId,taskId);
*/
```

C Solution:

```
/*
* Problem: Design a Todo List
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
typedef struct {

} TodoList;


TodoList* todoListCreate() {

}

int todoListAddTask(TodoList* obj, int userId, char* taskDescription, int
dueDate, char** tags, int tagsSize) {

}

char** todoListGetAllTasks(TodoList* obj, int userId, int* retSize) {

}

char** todoListGetTasksForTag(TodoList* obj, int userId, char* tag, int*
retSize) {

}

void todoListCompleteTask(TodoList* obj, int userId, int taskId) {

}

void todoListFree(TodoList* obj) {

}

/**
* Your TodoList struct will be instantiated and called as such:
* TodoList* obj = todoListCreate();
* int param_1 = todoListAddTask(obj, userId, taskDescription, dueDate, tags,
tagsSize);

* char** param_2 = todoListGetAllTasks(obj, userId, retSize);
* char** param_3 = todoListGetTasksForTag(obj, userId, tag, retSize);

* todoListCompleteTask(obj, userId, taskId);

* todoListFree(obj);
*/
```

Go Solution:

```go
// Problem: Design a Todo List
// Difficulty: Medium
// Tags: array, string, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type TodoList struct {

}



func Constructor() TodoList {

}



func (this *TodoList) AddTask(userId int, taskDescription string, dueDate
int, tags []string) int {

}



func (this *TodoList) GetAllTasks(userId int) []string {

}



func (this *TodoList) GetTasksForTag(userId int, tag string) []string {

}
```

```
func (this *TodoList) CompleteTask(userId int, taskId int) {

}


/**
* Your TodoList object will be instantiated and called as such:
* obj := Constructor();
* param_1 := obj.AddTask(userId,taskDescription,dueDate,tags);
* param_2 := obj.GetAllTasks(userId);
* param_3 := obj.GetTasksForTag(userId,tag);
* obj.CompleteTask(userId,taskId);
*/
```

Kotlin Solution:

```
class TodoList() {

fun addTask(userId: Int, taskDescription: String, dueDate: Int, tags:
List<String>): Int {

}

fun getAllTasks(userId: Int): List<String> {

}

fun getTasksForTag(userId: Int, tag: String): List<String> {

}

fun completeTask(userId: Int, taskId: Int) {

}

}

/**
* Your TodoList object will be instantiated and called as such:
```

```
* var obj = TodoList()
* var param_1 = obj.addTask(userId,taskDescription,dueDate,tags)
* var param_2 = obj.getAllTasks(userId)
* var param_3 = obj.getTasksForTag(userId,tag)
* obj.completeTask(userId,taskId)
*/
```

Swift Solution:

```
class TodoList {

init() {

}

func addTask(_ userId: Int, _ taskDescription: String, _ dueDate: Int, _
tags: [String]) -> Int {

}

func getAllTasks(_ userId: Int) -> [String] {

}

func getTasksForTag(_ userId: Int, _ tag: String) -> [String] {

}

func completeTask(_ userId: Int, _ taskId: Int) {

}
}

/**
* Your TodoList object will be instantiated and called as such:
* let obj = TodoList()
* let ret_1: Int = obj.addTask(userId, taskDescription, dueDate, tags)
* let ret_2: [String] = obj.getAllTasks(userId)
* let ret_3: [String] = obj.getTasksForTag(userId, tag)
* obj.completeTask(userId, taskId)
```

```
*/
```

Rust Solution:

```
// Problem: Design a Todo List
// Difficulty: Medium
// Tags: array, string, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

struct TodoList {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl TodoList {

fn new() -> Self {

}

fn add_task(&self, user_id: i32, task_description: String, due_date: i32,
tags: Vec<String>) -> i32 {

}

fn get_all_tasks(&self, user_id: i32) -> Vec<String> {

}

fn get_tasks_for_tag(&self, user_id: i32, tag: String) -> Vec<String> {

}

fn complete_task(&self, user_id: i32, task_id: i32) {
}
}

/**
* Your TodoList object will be instantiated and called as such:
* let obj = TodoList::new();
* let ret_1: i32 = obj.add_task(userId, taskDescription, dueDate, tags);
* let ret_2: Vec<String> = obj.get_all_tasks(userId);
* let ret_3: Vec<String> = obj.get_tasks_for_tag(userId, tag);
* obj.complete_task(userId, taskId);
*/
```

Ruby Solution:

```
class TodoList
def initialize()

end



=begin
:type user_id: Integer
:type task_description: String
:type due_date: Integer
:type tags: String[]
:rtype: Integer
=end
def add_task(user_id, task_description, due_date, tags)

end



=begin
:type user_id: Integer
:rtype: String[]
=end
def get_all_tasks(user_id)

end
```

```



=begin
:type user_id: Integer
:type tag: String
:rtype: String[]
=end
def get_tasks_for_tag(user_id, tag)

end



=begin
:type user_id: Integer
:type task_id: Integer
:rtype: Void
=end
def complete_task(user_id, task_id)

end



end

# Your TodoList object will be instantiated and called as such:
# obj = TodoList.new()
# param_1 = obj.add_task(user_id, task_description, due_date, tags)
# param_2 = obj.get_all_tasks(user_id)
# param_3 = obj.get_tasks_for_tag(user_id, tag)
# obj.complete_task(user_id, task_id)
```

PHP Solution:

```
class TodoList {
/**
*/
function __construct() {

}

/**
* @param Integer $userId
* @param String $taskDescription
* @param Integer $dueDate
* @param String[] $tags
* @return Integer
*/
function addTask($userId, $taskDescription, $dueDate, $tags) {

}

/**
* @param Integer $userId
* @return String[]
*/
function getAllTasks($userId) {

}

/**
* @param Integer $userId
* @param String $tag
* @return String[]
*/
function getTasksForTag($userId, $tag) {

}

/**
* @param Integer $userId
* @param Integer $taskId
* @return NULL
*/
function completeTask($userId, $taskId) {

}
}

/**
* Your TodoList object will be instantiated and called as such:
* $obj = TodoList();
* $ret_1 = $obj->addTask($userId, $taskDescription, $dueDate, $tags);
* $ret_2 = $obj->getAllTasks($userId);
* $ret_3 = $obj->getTasksForTag($userId, $tag);
* $obj->completeTask($userId, $taskId);
*/
```

Dart Solution:

```
class TodoList {

TodoList() {

}

int addTask(int userId, String taskDescription, int dueDate, List<String>
tags) {

}

List<String> getAllTasks(int userId) {

}

List<String> getTasksForTag(int userId, String tag) {

}

void completeTask(int userId, int taskId) {

}
}

/**
* Your TodoList object will be instantiated and called as such:
* TodoList obj = TodoList();
* int param1 = obj.addTask(userId,taskDescription,dueDate,tags);
* List<String> param2 = obj.getAllTasks(userId);
* List<String> param3 = obj.getTasksForTag(userId,tag);
* obj.completeTask(userId,taskId);
*/
```

Scala Solution:

```
class TodoList() {

def addTask(userId: Int, taskDescription: String, dueDate: Int, tags:
List[String]): Int = {

}

def getAllTasks(userId: Int): List[String] = {

}

def getTasksForTag(userId: Int, tag: String): List[String] = {

}

def completeTask(userId: Int, taskId: Int): Unit = {

}

}

/**
* Your TodoList object will be instantiated and called as such:
* val obj = new TodoList()
* val param_1 = obj.addTask(userId,taskDescription,dueDate,tags)
* val param_2 = obj.getAllTasks(userId)
* val param_3 = obj.getTasksForTag(userId,tag)
* obj.completeTask(userId,taskId)
*/
```

Elixir Solution:

```
defmodule TodoList do
@spec init_() :: any
def init_() do

end

@spec add_task(user_id :: integer, task_description :: String.t, due_date ::
integer, tags :: [String.t]) :: integer
def add_task(user_id, task_description, due_date, tags) do
```

```

end

@spec get_all_tasks(user_id :: integer) :: [String.t]
def get_all_tasks(user_id) do

end

@spec get_tasks_for_tag(user_id :: integer, tag :: String.t) :: [String.t]
def get_tasks_for_tag(user_id, tag) do

end

@spec complete_task(user_id :: integer, task_id :: integer) :: any
def complete_task(user_id, task_id) do

end
end

# Your functions will be called as such:
# TodoList.init_()
# param_1 = TodoList.add_task(user_id, task_description, due_date, tags)
# param_2 = TodoList.get_all_tasks(user_id)
# param_3 = TodoList.get_tasks_for_tag(user_id, tag)
# TodoList.complete_task(user_id, task_id)

# TodoList.init_ will be called before every test case, in which you can do
some necessary initializations.
```

Erlang Solution:

```
-spec todo_list_init_() -> any().
todo_list_init_() ->
.

-spec todo_list_add_task(UserId :: integer(), TaskDescription ::
unicode:unicode_binary(), DueDate :: integer(), Tags ::
[unicode:unicode_binary()]) -> integer().
todo_list_add_task(UserId, TaskDescription, DueDate, Tags) ->
.

-spec todo_list_get_all_tasks(UserId :: integer()) ->
```

```
[unicode:unicode_binary()].
todo_list_get_all_tasks(UserId) ->
.

-spec todo_list_get_tasks_for_tag(UserId :: integer(), Tag ::
unicode:unicode_binary()) -> [unicode:unicode_binary()].
todo_list_get_tasks_for_tag(UserId, Tag) ->
.

-spec todo_list_complete_task(UserId :: integer(), TaskId :: integer()) ->
any().
todo_list_complete_task(UserId, TaskId) ->
.



%% Your functions will be called as such:
%% todo_list_init_(),
%% Param_1 = todo_list_add_task(UserId, TaskDescription, DueDate, Tags),
%% Param_2 = todo_list_get_all_tasks(UserId),
%% Param_3 = todo_list_get_tasks_for_tag(UserId, Tag),
%% todo_list_complete_task(UserId, TaskId),

%% todo_list_init_ will be called before every test case, in which you can do
some necessary initializations.
```

Racket Solution:

```
(define todo-list%
(class object%
(super-new)

(init-field)

; add-task : exact-integer? string? exact-integer? (listof string?) ->
exact-integer?
(define/public (add-task user-id task-description due-date tags)
)
; get-all-tasks : exact-integer? -> (listof string?)
(define/public (get-all-tasks user-id)
)
; get-tasks-for-tag : exact-integer? string? -> (listof string?)
(define/public (get-tasks-for-tag user-id tag)
)
; complete-task : exact-integer? exact-integer? -> void?
(define/public (complete-task user-id task-id)
)))

;; Your todo-list% object will be instantiated and called as such:
;; (define obj (new todo-list%))
;; (define param_1 (send obj add-task user-id task-description due-date
tags))
;; (define param_2 (send obj get-all-tasks user-id))
;; (define param_3 (send obj get-tasks-for-tag user-id tag))
;; (send obj complete-task user-id task-id)
```