COMPONENT A: PROGRAM CODE

Submit one PDF file that contains all of your program code (including comments). Include comments or acknowledgments for any part of the submitted program code that has been written by someone other than you.

Important: With text-based program. code, you can use the print command to save your program code as a PDF file, or you can copy and paste your code to a text document and then convert it into a PDF file. Screen captures should not be blurry, and text should be at least 10 pt font size.

- ☐ In your program, you must include student-developed program code that contains the following:
  Instructions for input from one of the following:
  - ◆ the user (including user actions that trigger events)
  - ◆ a device
  - ◆ an online data stream
  - ◆ a file

- ☐ Use of at least one list (or other collection type) to represent a collection of data that is stored and used to manage program complexity and help fulfill the program's purpose

Important: The data abstraction must make the program easier to develop (alternatives would be more complex) or easier to maintain (future changes to the size of the list would otherwise require significant modifications to the program code)

- ☐ At least one procedure that contributes to the program's intended purpose, where you have defined:
  - ◆ the procedure's name
  - ◆ the return type (if necessary)
  - ◆ one or more parameters

- ☐ An algorithm that includes sequencing, selection, and iteration that is in the body of the selected procedure
- ☐ Calls to your student-developed procedure
- ☐ Instructions for output (tactile, audible, visual, or textual) based on input and program functionality

COMPONENT B: VIDEO (CREATED INDEPENDENTLY)

Submit one video file that demonstrates the running of your program as described below. Collaboration is not allowed during the development of your video.

Your video must demonstrate your program running, including:

- ☐ Input to your program
- ☐ At least one aspect of the functionality of your program
- ☐ Output produced by your program

Your video may NOT contain:

- ☐ Any distinguishing information about yourself
- ☐ Voice narration (though text captions are encouraged)

Your video must be:

- ☐ ◻ Either .webm, .mp4, .wmv, .avi, or .mov format
- ☐ ◻ No more than 1 minute in length
- ☐ ◻ No more than 30MB in file size

COMPONENT C: PERSONALIZED PROJECT REFERENCE

Submit your responses to prompts 3a – 3d, which are described below. Your response to all prompts combined must not exceed 750 words (program code is not included in the word count).. Instructions for submitting your written responses are available on the **AP Computer Science Principles Exam Page** on AP Central.

3 a. Provide a written response that does all three of the following:
Approx. 150 words (for all subparts of 3a combined)

i. Describes the overall purpose of the program

My program is a Simple Day Planner app designed to help users organize their daily tasks. It allows users to create, view, complete, and delete tasks with specific due dates. Users can easily filter tasks by date or search for specific tasks using keywords, making it a practical tool for personal task management.

ii. Describes what functionality of the program is demonstrated in the video

The video demonstrates the core features of my Day Planner app, including adding a new task with a title and due date, viewing tasks for a selected date, marking a task as completed (which shows a visual strikethrough effect), and using the search feature to find specific tasks by keyword.

iii. Describes the input and output of the program demonstrated in the video

For input, my program accepts user interactions through the UI including entering task titles and due dates when creating tasks, selecting dates from a date picker to view specific days, entering search keywords, and button interactions to mark tasks complete or delete them. The output includes a visually formatted list of tasks that can be filtered by date or search term, with visual indicators for completion status (checkmarks and strikethrough text) and feedback when no tasks are found.

3 b. Capture and paste two program code segments you developed during the administration of this task that contain a list (or other collection type) being used to manage complexity in your program.
Approx. 200 words (for all subparts of 3b combined, exclusive of program code)

i. The first program code segment must show how data have been stored in the list.

```
// This class manages our collection of tasks
class TaskManager: ObservableObject {
    // This is our list collection that stores all tasks
    @Published var tasks: [Task] = []

    // Add a new task to our collection
    func addTask(title: String, dueDate: Date) {
        let newTask = Task(id: UUID(), title: title, isCompleted: false, dueDate:
dueDate)
        tasks.append(newTask)
    }
}
```

DEFINITION:
List
A **list** is an ordered sequence of elements. The use of lists allows multiple related items to be represented using a single variable. Lists may be referred to by different names, such as arrays, depending on the programming language.

Then, provide a written response that does all three of the following:
iii. Identifies the name of the list being used in this response

The list in my program is named `tasks`.

IMPORTANT:
The data abstraction must make the program easier to develop (alternatives would be more complex) or easier to maintain (future changes to the size of the list would otherwise require significant modifications to the

iv. Describes what the data contained in the list represent in your program

The `tasks` list contains Task objects that represent user-created activities or to-dos. Each Task has properties including a unique identifier (id), a descriptive title, a boolean completion status, and a due date. This collection essentially represents all the tasks that a user needs to track in their planner.

v. Explains how the selected list manages complexity in your program code by explaining why your program code could not be written, or how it would be written differently, if you did not use the list

IMPORTANT: Built-in or existing procedures and language structures, such as event handlers and main methods, are not considered studentdeveloped.

Using a list to store tasks dramatically reduces the complexity of my program. Without this list, I would need separate variables for each task, making it impossible to handle an unknown number of tasks. The list allows me to dynamically add and remove tasks at runtime without changing the code structure. It enables operations like filtering by date, searching by keyword, and batch processing which would be extremely difficult otherwise. The list abstraction also simplifies the UI code since I can iterate through the collection to display tasks.

3c. Capture and paste two program code segments you developed during the administration of this task that contain a student-developed procedure that implements an algorithm used in your program and a call to that procedure.

Approx. 200 words (for all subparts of 3c combined, exclusive of program code)

i. The first program code segment must be a student-developed procedure that:

□ Defines the procedure's name and return type (if necessary)

□ Contains and uses one or more parameters that have an effect on the functionality of the procedure

□ Implements an algorithm that includes sequencing, selection, and iteration

```
// This procedure filters tasks by a search term
// Parameters: keyword - the search term to look for
// Returns: a filtered list of tasks matching the keyword
func searchTasks(keyword: String) -> [Task] {
    // Create empty result list
    var results: [Task] = []
    // Iterate through all tasks to find matches
    for task in tasks {
        // Selection: Check if task title contains the keyword
        if task.title.lowercased().contains(keyword.lowercased()) {
            // Add matching task to results
            results.append(task)
        }
    }
    // Return the filtered collection
    return results
}
```

ii. The second program code segment must show where your student-developed procedure is being called in your program.

```
// Determine which tasks to show (searched or date-filtered)
let displayedTasks = isSearching ?
    taskManager.searchTasks(keyword: searchText) :
    tasksForSelectedDate()
```

Then, provide a written response that does both of the following:

iii. Describes in general what the identified procedure does and how it contributes to the overall functionality of the program

The `searchTasks` procedure I developed filters the task list based on a user-provided search keyword. This function is crucial to the app's functionality because it allows users to quickly find relevant tasks by searching for words in the task titles instead of scrolling through their entire task list or browsing by date. This greatly improves the user experience, especially as the number of tasks grows over time.

iv. Explains in detailed steps how the algorithm implemented in the identified procedure works. Your explanation must be detailed enough for someone else to recreate it.

My search algorithm works through these detailed steps:

1. It takes a string parameter `keyword` which is the search term entered by the user
2. It creates an empty array called `results` to store any matching tasks
3. It loops through each task in the main tasks collection
4. For each task, it converts both the task title and search keyword to lowercase using

`.lowercased()` to make the search case-insensitive

5. It uses the `.contains()` string method to check if the lowercase task title includes the lowercase search term
6. When a match is found, it adds that task to the results array using `.append()`
7. After checking all tasks, it returns the results array with only the matching tasks

3d. Provide a written response that does all three of the following:

Approx. 200 words (for all subparts of 3d combined)

i. Describes two calls to the procedure identified in written response 3c. Each call must pass a different argument(s) that causes a different segment of code in the algorithm to execute.

First call:

When a users enter homework
taskManager.searchTasks(keyword: "homework")

Second call:

When a user clears the search box (empty string), the program calls:

```
Unset

taskManager.searchTasks(keyword: "")
```

ii. Describes what condition(s) is being tested by each call to the procedure

Condition(s) tested by the first call:

The first call tests whether any task titles contain the word "homework" (case-insensitive). The procedure checks if the lowercase version of each task title contains the lowercase "homework" string.

Condition(s) tested by the second call:

The second call tests if task titles contain an empty string. Since any string contains an empty string (even if it's just at the beginning or end), this condition will be true for all tasks.

iii. Identifies the result of each call

Result of the first call:

The function returns a filtered list containing only tasks with "homework" in their titles. For example, tasks like "Math homework", "Finish homework", or "Science homework due" would be included in the results. If no matching tasks exist, it returns an empty array and the UI displays "No tasks found".

Result of the second call:

The function returns a list containing all tasks from the original collection, since every string contains an empty string. This effectively shows an unfiltered list, the same as if the user wasn't searching at all.