WOOSONG UNIVERSITY

ENDICOTT COLLEGE OF INTERNATIONAL STUDIES

# Final Project Presentation

## Operating Systems by PBL

# PBL - Team

- ✓ Class: Operating System (001)
- ✓ Professor: Young Il Kim
- ✓ **Team Organization**

| **PBL Name** | **Secure the Quality of OS in Space Shuttle** | |
|---|---|---|
| **Team Name** | Corner Crew | |
| **Team Member** | Leader | Cavan Schutte |
| **Role & Responsibility**<br>*(This is sub-R&R. Main R&R of each member is to write test programs)* | Presentation | Victor Oluwaseun Opaleke |
| | Information Search | 조윤주 |
| | Meeting Note<br>& Report Writer | 차동후 |
| | Presentation Maker | Heang Seavleu |
| | Demonstration | Cavan Schutte |

# Content

**1- Project Introduction:** Project Goal, Key Achievements, and Limitation

**2- Summary of Project:** Project Purpose, Background, Scope, Milestone, and Methodology

**3- Project Execution**

- Test execution plan
- Decision behind the chosen methods
- How to solve the design problem
- How the purpose of the project is accomplished in the final design & execution?

**4- Conclusion and Recommendation**

- Describe conclusions and suggestions
- Expected outcome if given an additional two weeks

# Project Instruction

## ✓ Project Goal

- Test as many APIs as we could. (Tested 40 in total)

- Test each APIs with the simplest and effective way as we possibly could

- Bonus (but didn’t have time) - measure performance metrics of each APIs such as execution time, memory usage, and CPU utilization.

- Testing each API with incorrect parameters
    - It was challenging to just get the API to work, so this is always easy

# Project Instruction

## ✓ Project Goal (Continue)

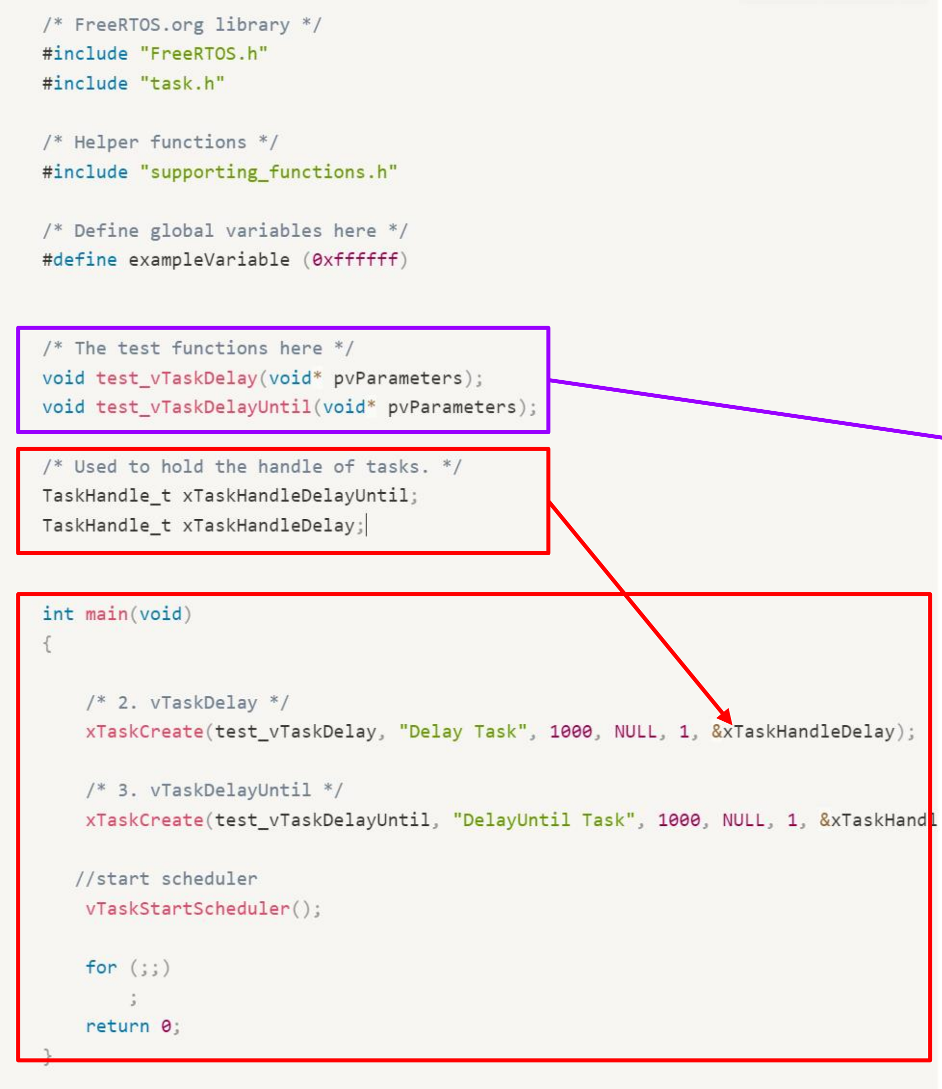
- Generate a Quality Report for the APIs: including usage instruction and demonstration.

- Project Closure and Handover: Conduct final review of the project outcome by ensuring all the objectives are delivered and met

# Project Instruction

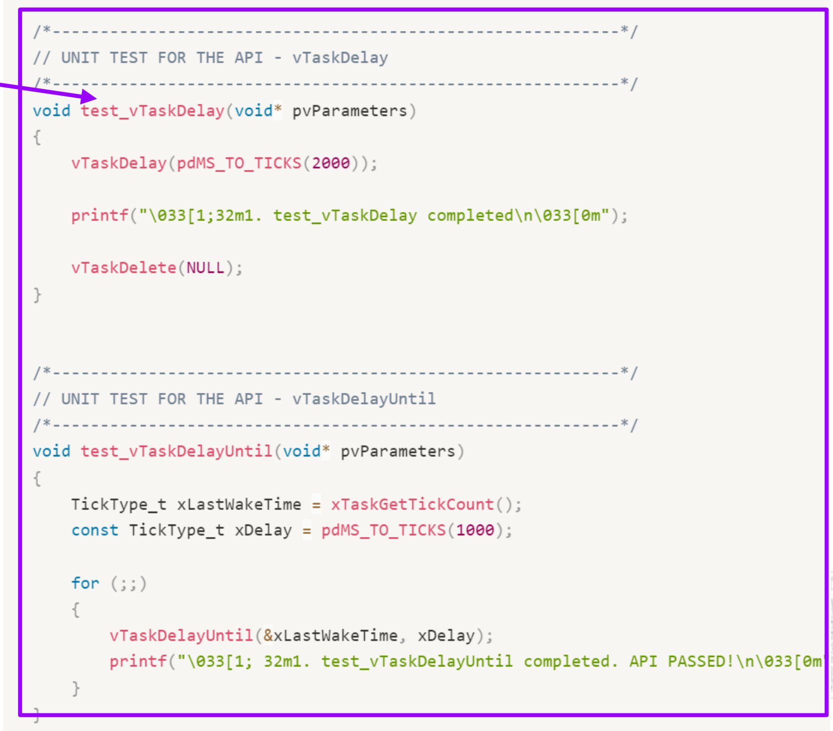
Corner Crew

## ✓ Key Achievements - Developed a Blinky File (Before)



```
/* FreeRTOS.org library */
#include "FreeRTOS.h"
#include "task.h"

/* Helper functions */
#include "supporting_functions.h"

/* Define global variables here */
#define exampleVariable (0xffffff)


/* The test functions here */
void test_vTaskDelay(void* pvParameters);
void test_vTaskDelayUntil(void* pvParameters);

/* Used to hold the handle of tasks. */
TaskHandle_t xTaskHandleDelayUntil;
TaskHandle_t xTaskHandleDelay;


int main(void)
{

    /* 2. vTaskDelay */
    xTaskCreate(test_vTaskDelay, "Delay Task", 1000, NULL, 1, &xTaskHandleDelay);

    /* 3. vTaskDelayUntil */
    xTaskCreate(test_vTaskDelayUntil, "DelayUntil Task", 1000, NULL, 1, &xTaskHandl

   //start scheduler
    vTaskStartScheduler();

    for (;;)
        ;
    return 0;
}
```

- Too many test function name!🤯
- Time-consuming and unorganised😫

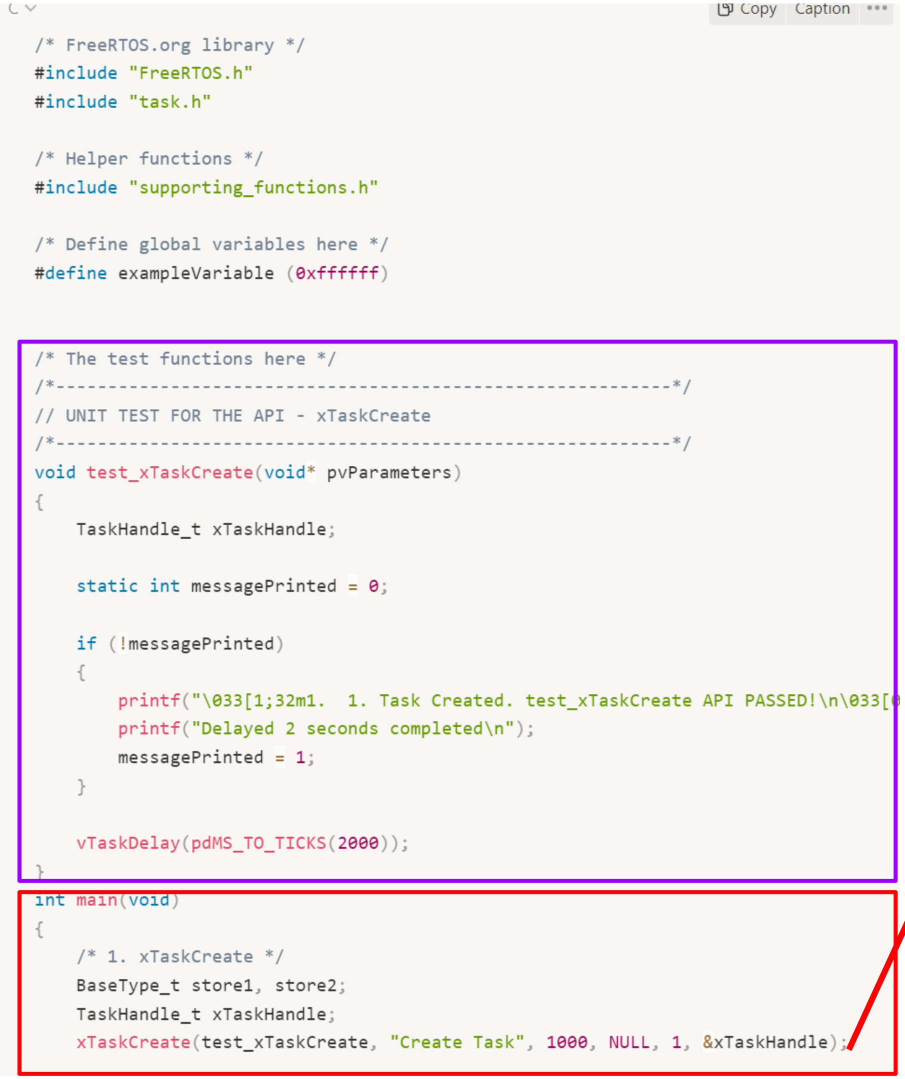


```
/*-----------------------------------------------------------------*/
// UNIT TEST FOR THE API - vTaskDelay
/*-----------------------------------------------------------------*/
void test_vTaskDelay(void* pvParameters)
{
    vTaskDelay(pdMS_TO_TICKS(2000));

    printf("\033[1;32m1. test_vTaskDelay completed\n\033[0m");

    vTaskDelete(NULL);
}


/*-----------------------------------------------------------------*/
// UNIT TEST FOR THE API - vTaskDelayUntil
/*-----------------------------------------------------------------*/
void test_vTaskDelayUntil(void* pvParameters)
{
    TickType_t xLastWakeTime = xTaskGetTickCount();
    const TickType_t xDelay = pdMS_TO_TICKS(1000);

    for (;;)
    {
        vTaskDelayUntil(&xLastWakeTime, xDelay);
        printf("\033[1; 32m1. test_vTaskDelayUntil completed. API PASSED!\n\033[0m
    }
}
```

# Project Instruction

Corner Crew

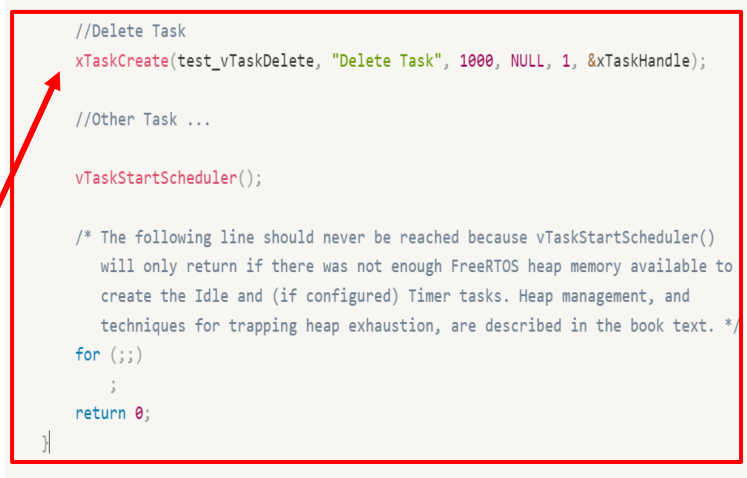
## ✓ **Key Achievements -** Developed a Blinky File **(After)**



```c
/* FreeRTOS.org library */
#include "FreeRTOS.h"
#include "task.h"

/* Helper functions */
#include "supporting_functions.h"

/* Define global variables here */
#define exampleVariable (0xffffff)


/* The test functions here */
/*-----------------------------------------------------------*/
// UNIT TEST FOR THE API - xTaskCreate
/*-----------------------------------------------------------*/
void test_xTaskCreate(void* pvParameters)
{
    TaskHandle_t xTaskHandle;

    static int messagePrinted = 0;

    if (!messagePrinted)
    {
        printf("\033[1;32m1.  1. Task Created. test_xTaskCreate API PASSED!\n\033[0
        printf("Delayed 2 seconds completed\n");
        messagePrinted = 1;
    }

    vTaskDelay(pdMS_TO_TICKS(2000));
}
int main(void)
{
    /* 1. xTaskCreate */
    BaseType_t store1, store2;
    TaskHandle_t xTaskHandle;
    xTaskCreate(test_xTaskCreate, "Create Task", 1000, NULL, 1, &xTaskHandle);
```

- No need to define function name
- Code will run without error (if there's no bugs)

As a result, this structure allows us to achieve a more clean and simple to understand code



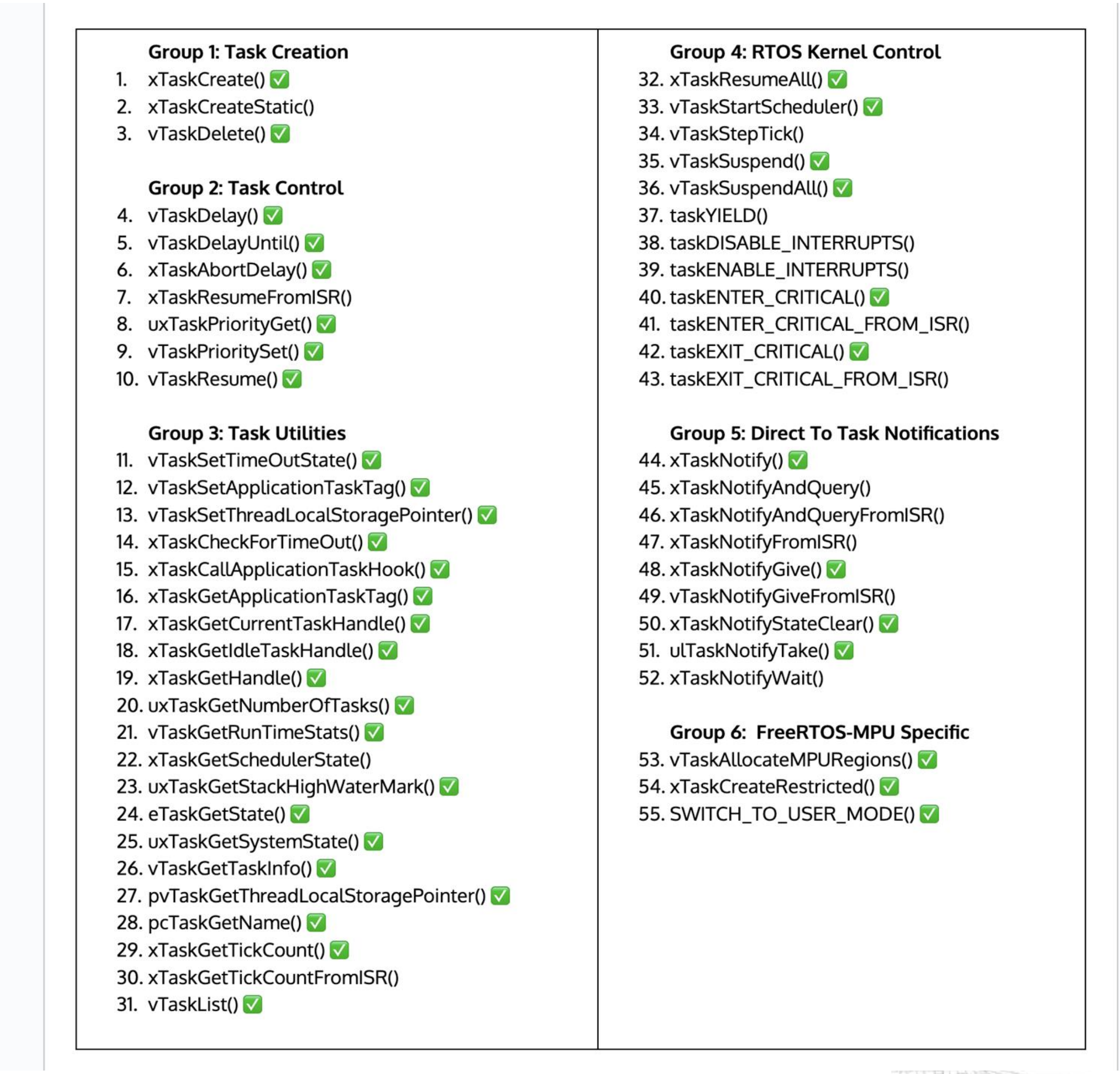
```c
    //Delete Task
    xTaskCreate(test_vTaskDelete, "Delete Task", 1000, NULL, 1, &xTaskHandle);

    //Other Task ...

    vTaskStartScheduler();

    /* The following line should never be reached because vTaskStartScheduler()
       will only return if there was not enough FreeRTOS heap memory available to
       create the Idle and (if configured) Timer tasks. Heap management, and
       techniques for trapping heap exhaustion, are described in the book text. */
    for (;;)
        ;
    return 0;
}
```

# Project Instruction

Corner Crew

## ✓ **Key Achievements -** 40 APIs are tested

- 40 APIs implemented successfully
- Some do not work due to configuration issues



**Group 1: Task Creation**

1. xTaskCreate() ✅
2. xTaskCreateStatic()
3. vTaskDelete() ✅

**Group 2: Task Control**

4. vTaskDelay() ✅
5. vTaskDelayUntil() ✅
6. xTaskAbortDelay() ✅
7. xTaskResumeFromISR()
8. uxTaskPriorityGet() ✅
9. vTaskPrioritySet() ✅
10. vTaskResume() ✅

**Group 3: Task Utilities**

11. vTaskSetTimeOutState() ✅
12. vTaskSetApplicationTaskTag() ✅
13. vTaskSetThreadLocalStoragePointer() ✅
14. xTaskCheckForTimeOut() ✅
15. xTaskCallApplicationTaskHook() ✅
16. xTaskGetApplicationTaskTag() ✅
17. xTaskGetCurrentTaskHandle() ✅
18. xTaskGetIdleTaskHandle() ✅
19. xTaskGetHandle() ✅
20. uxTaskGetNumberOfTasks() ✅
21. vTaskGetRunTimeStats() ✅
22. xTaskGetSchedulerState()
23. uxTaskGetStackHighWaterMark() ✅
24. eTaskGetState() ✅
25. uxTaskGetSystemState() ✅
26. vTaskGetTaskInfo() ✅
27. pvTaskGetThreadLocalStoragePointer() ✅
28. pcTaskGetName() ✅
29. xTaskGetTickCount() ✅
30. xTaskGetTickCountFromISR()
31. vTaskList() ✅

**Group 4: RTOS Kernel Control**

32. xTaskResumeAll() ✅
33. vTaskStartScheduler() ✅
34. vTaskStepTick()
35. vTaskSuspend() ✅
36. vTaskSuspendAll() ✅
37. taskYIELD()
38. taskDISABLE_INTERRUPTS()
39. taskENABLE_INTERRUPTS()
40. taskENTER_CRITICAL() ✅
41. taskENTER_CRITICAL_FROM_ISR()
42. taskEXIT_CRITICAL() ✅
43. taskEXIT_CRITICAL_FROM_ISR()

**Group 5: Direct To Task Notifications**

44. xTaskNotify() ✅
45. xTaskNotifyAndQuery()
46. xTaskNotifyAndQueryFromISR()
47. xTaskNotifyFromISR()
48. xTaskNotifyGive() ✅
49. vTaskNotifyGiveFromISR()
50. xTaskNotifyStateClear() ✅
51. ulTaskNotifyTake() ✅
52. xTaskNotifyWait()

**Group 6: FreeRTOS-MPU Specific**

53. vTaskAllocateMPURegions() ✅
54. xTaskCreateRestricted() ✅
55. SWITCH_TO_USER_MODE() ✅

# Project Instruction

## ✓ **Key Achievements** - Generate the Quality Report

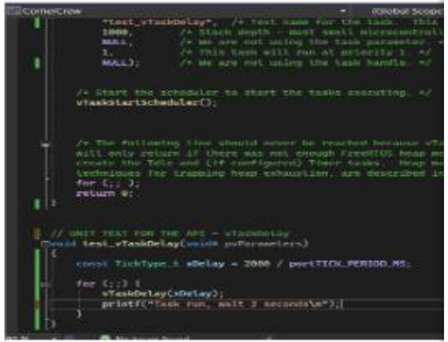
- Each API includes
  - Was it successfully executed
  - Sample of the terminal output
  - Snippet of the code

## Group 2: Task Cont

vTaskDelay()

- Executed successfully: ✅

- Demonstration



```
        "test_vTaskDelay",  /* Text name for the task.  This
        1000,         /* Stack depth - most small microcontroll
        NULL,         /* We are not using the task parameter.
        1,            /* This task will run at priority 1.  */
        NULL);        /* We are not using the task handle.  */

    /* Start the scheduler to start the tasks executing.  */
    vTaskStartScheduler();

    /* The following line should never be reached because vTa
    will only return if there was not enough FreeRTOS heap me
    create the Idle and (if configured) Timer tasks.  Heap ma
    techniques for trapping heap exhaustion, are described in
    for (;; );
    return 0;
}

// UNIT TEST FOR THE API - vTaskDelay
void test_vTaskDelay(void* pvParameters)
{
    const TickType_t xDelay = 2000 / portTICK_PERIOD_MS;

    for (;;) {
        vTaskDelay(xDelay);
        printf("Task run, wait 2 seconds\n");
    }
}
```

vTaskDelayUntil()

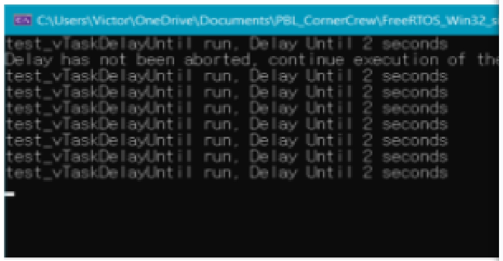
- Executed successfully: ✅

- Demonstration

xTaskAbortDelay()

- Executed successfully: ✅

- Demonstration



C:\Users\Victor\OneDrive\Documents\PBL_CornerCrew\FreeRTOS_Win32_s

```
test_vTaskDelayUntil run, Delay Until 2 seconds
Delay has not been aborted, continue execution of the
test_vTaskDelayUntil run, Delay Until 2 seconds
test_vTaskDelayUntil run, Delay Until 2 seconds
test_vTaskDelayUntil run, Delay Until 2 seconds
test_vTaskDelayUntil run, Delay Until 2 seconds
test_vTaskDelayUntil run, Delay Until 2 seconds
test_vTaskDelayUntil run, Delay Until 2 seconds
test_vTaskDelayUntil run, Delay Until 2 seconds
```

xTaskResumeFromISR()

- Executed successfully: ✅ ❌

- Demonstration

uxTaskPriorityGet()

- Executed successfully: ✅

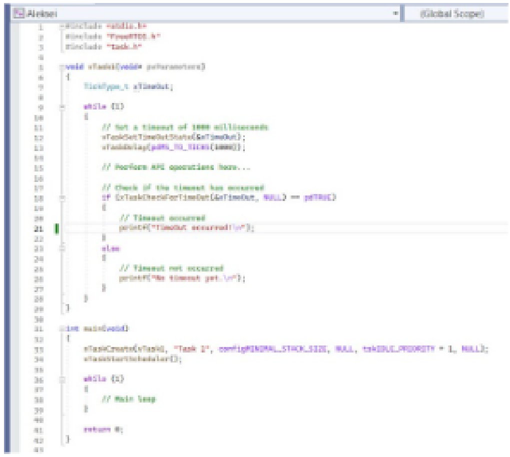
- Demonstration

## Group 3: Task Utilities

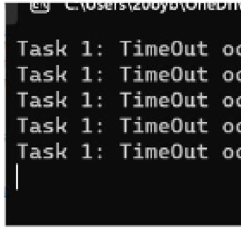
vTaskSetTimeOutState()

- Executed successfully: ✅

- Demonstration



```
#include <stdio.h>
#include "FreeRTOS.h"
#include "task.h"

void vTask1(void* pvParameters)
{
    TickType_t xTimeOut;

    while (1)
    {
        // Set a timeout of 1000 milliseconds
        vTaskSetTimeOutState(&xTimeOut);
        vTaskDelay(pdMS_TO_TICKS(1000));

        // Perform API operations here...

        // Check if the timeout has occurred
        if (xTaskCheckForTimeOut(&xTimeOut, NULL) == pdTRUE)
        {
            // Timeout occurred
            printf("TimeOut occurred!\n");
        }
        else
        {
            // Timeout not occurred
            printf("No timeout yet.\n");
        }
    }
}

int main(void)
{
    xTaskCreate(vTask1, "Task 1", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY + 1, NULL);
    vTaskStartScheduler();

    while (1)
    {
        // Main loop
    }

    return 0;
}
```



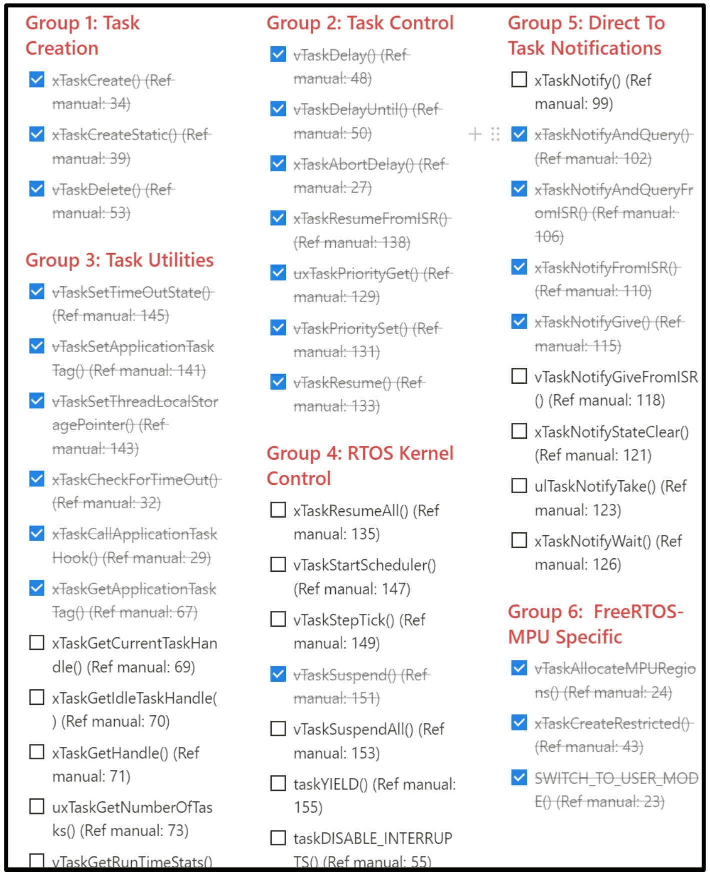
C:\Users\zooyb\OneDri

```
Task 1: TimeOut oc
Task 1: TimeOut oc
Task 1: TimeOut oc
Task 1: TimeOut oc
Task 1: TimeOut oc
```

vTaskSetApplicationTaskTag()

## ✓ Key Achievements (Continue)

•Behind the scene we invest lots of effort in both organizing and testing the API. This is the organised already-divided API based on the official APIs' categories



**Group 1: Task Creation**

- ☑ ~~xTaskCreate() (Ref manual: 34)~~
- ☑ ~~xTaskCreateStatic() (Ref manual: 39)~~
- ☑ ~~vTaskDelete() (Ref manual: 53)~~

**Group 3: Task Utilities**

- ☑ ~~vTaskSetTimeOutState() (Ref manual: 145)~~
- ☑ ~~vTaskSetApplicationTaskTag() (Ref manual: 141)~~
- ☑ ~~vTaskSetThreadLocalStoragePointer() (Ref manual: 143)~~
- ☑ ~~xTaskCheckForTimeOut() (Ref manual: 32)~~
- ☑ ~~xTaskCallApplicationTaskHook() (Ref manual: 29)~~
- ☑ ~~xTaskGetApplicationTaskTag() (Ref manual: 67)~~
- ☐ xTaskGetCurrentTaskHandle() (Ref manual: 69)
- ☐ xTaskGetIdleTaskHandle() (Ref manual: 70)
- ☐ xTaskGetHandle() (Ref manual: 71)
- ☐ uxTaskGetNumberOfTasks() (Ref manual: 73)
- ☐ vTaskGetRunTimeStats()

**Group 2: Task Control**

- ☑ ~~vTaskDelay() (Ref manual: 48)~~
- ☑ ~~vTaskDelayUntil() (Ref manual: 50)~~
- ☑ ~~xTaskAbortDelay() (Ref manual: 27)~~
- ☑ ~~xTaskResumeFromISR() (Ref manual: 138)~~
- ☑ ~~uxTaskPriorityGet() (Ref manual: 129)~~
- ☑ ~~vTaskPrioritySet() (Ref manual: 131)~~
- ☑ ~~vTaskResume() (Ref manual: 133)~~

**Group 4: RTOS Kernel Control**

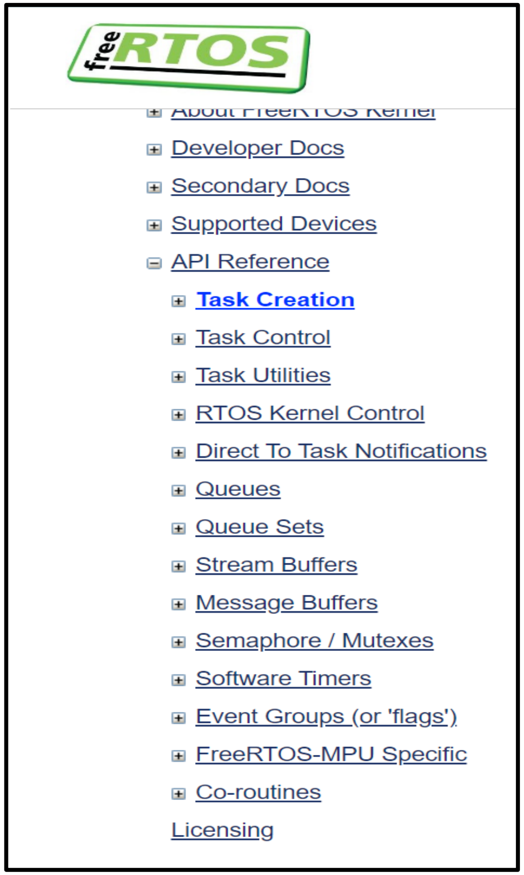
- ☐ xTaskResumeAll() (Ref manual: 135)
- ☐ vTaskStartScheduler() (Ref manual: 147)
- ☐ vTaskStepTick() (Ref manual: 149)
- ☑ ~~vTaskSuspend() (Ref manual: 151)~~
- ☐ vTaskSuspendAll() (Ref manual: 153)
- ☐ taskYIELD() (Ref manual: 155)
- ☐ taskDISABLE_INTERRUPTS() (Ref manual: 55)

**Group 5: Direct To Task Notifications**

- ☐ xTaskNotify() (Ref manual: 99)
- ☑ ~~xTaskNotifyAndQuery() (Ref manual: 102)~~
- ☑ ~~xTaskNotifyAndQueryFromISR() (Ref manual: 106)~~
- ☑ ~~xTaskNotifyFromISR() (Ref manual: 110)~~
- ☑ ~~xTaskNotifyGive() (Ref manual: 115)~~
- ☐ vTaskNotifyGiveFromISR() (Ref manual: 118)
- ☐ xTaskNotifyStateClear() (Ref manual: 121)
- ☐ ulTaskNotifyTake() (Ref manual: 123)
- ☐ xTaskNotifyWait() (Ref manual: 126)

**Group 6: FreeRTOS-MPU Specific**

- ☑ ~~vTaskAllocateMPURegions() (Ref manual: 24)~~
- ☑ ~~xTaskCreateRestricted() (Ref manual: 43)~~
- ☑ ~~SWITCH_TO_USER_MODE() (Ref manual: 23)~~



freeRTOS

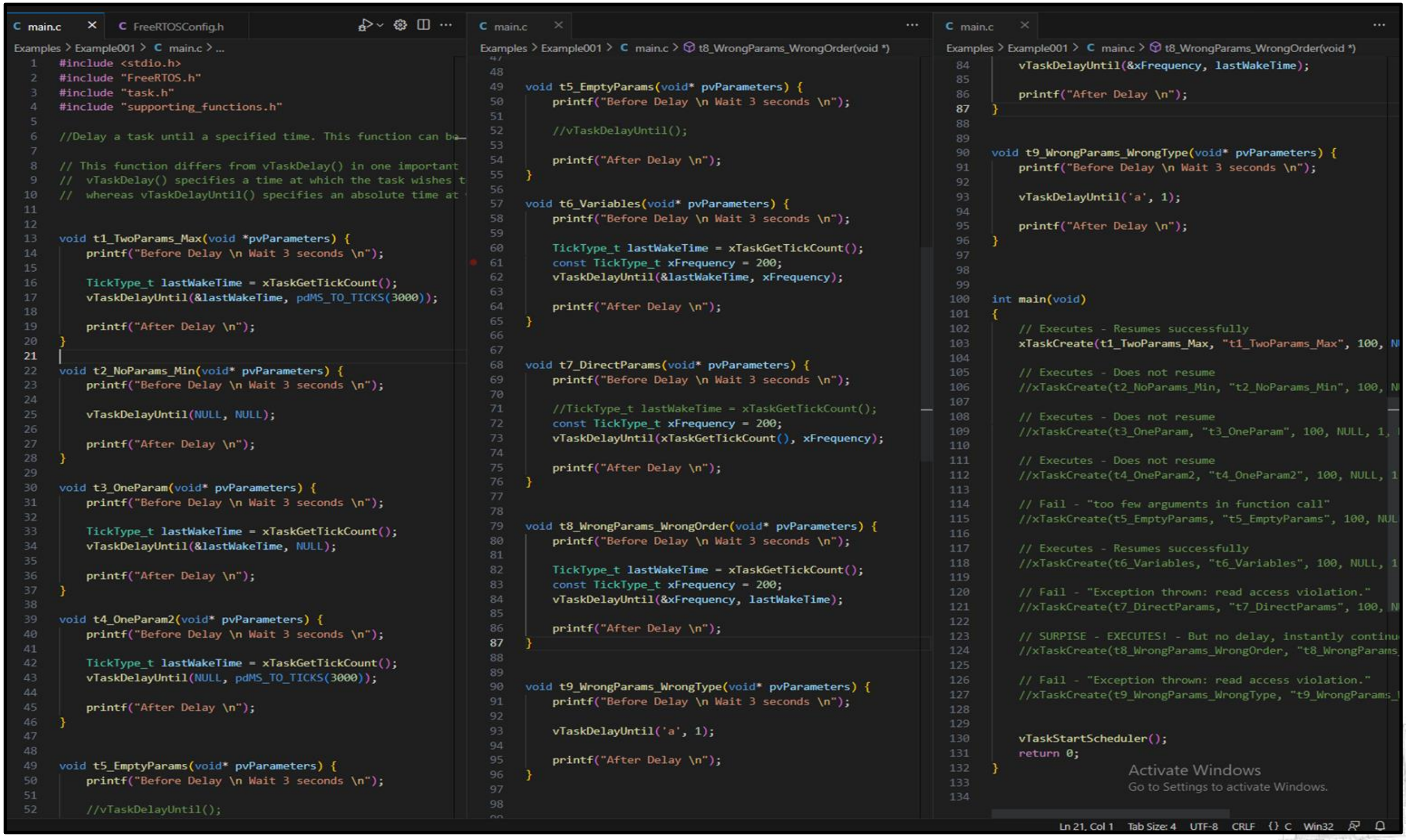
- ⊞ About FreeRTOS Kernel
- ⊞ Developer Docs
- ⊞ Secondary Docs
- ⊞ Supported Devices
- ⊟ API Reference
  - ⊞ **Task Creation**
  - ⊞ Task Control
  - ⊞ Task Utilities
  - ⊞ RTOS Kernel Control
  - ⊞ Direct To Task Notifications
  - ⊞ Queues
  - ⊞ Queue Sets
  - ⊞ Stream Buffers
  - ⊞ Message Buffers
  - ⊞ Semaphore / Mutexes
  - ⊞ Software Timers
  - ⊞ Event Groups (or 'flags')
  - ⊞ FreeRTOS-MPU Specific
  - ⊞ Co-routines
- Licensing

# Project Instruction

Corner Crew

## ✓ Key Achievements (Continue)

- Create a Demo for Demonstration (Will demo after this 😉)


main.c
FreeRTOSConfig.h
Examples > Example001 > main.c > ...
main.c
Examples > Example001 > main.c > t8_WrongParams_WrongOrder(void *)
main.c
Examples > Example001 > main.c > t8_WrongParams_WrongOrder(void *)
Activate Windows
Go to Settings to activate Windows.
Ln 21, Col 1
Tab Size: 4
UTF-8
CRLF
{} C
Win32

```c
#include <stdio.h>
#include "FreeRTOS.h"
#include "task.h"
#include "supporting_functions.h"

//Delay a task until a specified time. This function can be

// This function differs from vTaskDelay() in one important
//  vTaskDelay() specifies a time at which the task wishes t
//  whereas vTaskDelayUntil() specifies an absolute time at


void t1_TwoParams_Max(void *pvParameters) {
    printf("Before Delay \n Wait 3 seconds \n");

    TickType_t lastWakeTime = xTaskGetTickCount();
    vTaskDelayUntil(&lastWakeTime, pdMS_TO_TICKS(3000));

    printf("After Delay \n");
}

void t2_NoParams_Min(void* pvParameters) {
    printf("Before Delay \n Wait 3 seconds \n");

    vTaskDelayUntil(NULL, NULL);

    printf("After Delay \n");
}

void t3_OneParam(void* pvParameters) {
    printf("Before Delay \n Wait 3 seconds \n");

    TickType_t lastWakeTime = xTaskGetTickCount();
    vTaskDelayUntil(&lastWakeTime, NULL);

    printf("After Delay \n");
}

void t4_OneParam2(void* pvParameters) {
    printf("Before Delay \n Wait 3 seconds \n");

    TickType_t lastWakeTime = xTaskGetTickCount();
    vTaskDelayUntil(NULL, pdMS_TO_TICKS(3000));

    printf("After Delay \n");
}


void t5_EmptyParams(void* pvParameters) {
    printf("Before Delay \n Wait 3 seconds \n");

    //vTaskDelayUntil();
```

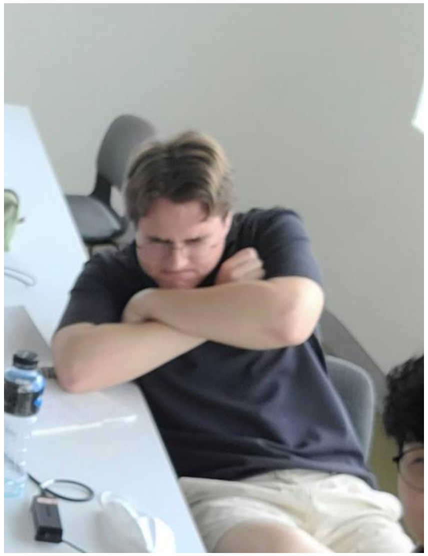
```c

void t5_EmptyParams(void* pvParameters) {
    printf("Before Delay \n Wait 3 seconds \n");

    //vTaskDelayUntil();

    printf("After Delay \n");
}

void t6_Variables(void* pvParameters) {
    printf("Before Delay \n Wait 3 seconds \n");

    TickType_t lastWakeTime = xTaskGetTickCount();
    const TickType_t xFrequency = 200;
    vTaskDelayUntil(&lastWakeTime, xFrequency);

    printf("After Delay \n");
}


void t7_DirectParams(void* pvParameters) {
    printf("Before Delay \n Wait 3 seconds \n");

    //TickType_t lastWakeTime = xTaskGetTickCount();
    const TickType_t xFrequency = 200;
    vTaskDelayUntil(xTaskGetTickCount(), xFrequency);

    printf("After Delay \n");
}


void t8_WrongParams_WrongOrder(void* pvParameters) {
    printf("Before Delay \n Wait 3 seconds \n");

    TickType_t lastWakeTime = xTaskGetTickCount();
    const TickType_t xFrequency = 200;
    vTaskDelayUntil(&xFrequency, lastWakeTime);

    printf("After Delay \n");
}


void t9_WrongParams_WrongType(void* pvParameters) {
    printf("Before Delay \n Wait 3 seconds \n");

    vTaskDelayUntil('a', 1);

    printf("After Delay \n");
}


```

```c
    vTaskDelayUntil(&xFrequency, lastWakeTime);

    printf("After Delay \n");
}


void t9_WrongParams_WrongType(void* pvParameters) {
    printf("Before Delay \n Wait 3 seconds \n");

    vTaskDelayUntil('a', 1);

    printf("After Delay \n");
}



int main(void)
{
    // Executes - Resumes successfully
    xTaskCreate(t1_TwoParams_Max, "t1_TwoParams_Max", 100, N

    // Executes - Does not resume
    //xTaskCreate(t2_NoParams_Min, "t2_NoParams_Min", 100, N

    // Executes - Does not resume
    //xTaskCreate(t3_OneParam, "t3_OneParam", 100, NULL, 1,

    // Executes - Does not resume
    //xTaskCreate(t4_OneParam2, "t4_OneParam2", 100, NULL, 1

    // Fail - "too few arguments in function call"
    //xTaskCreate(t5_EmptyParams, "t5_EmptyParams", 100, NUL

    // Executes - Resumes successfully
    //xTaskCreate(t6_Variables, "t6_Variables", 100, NULL, 1

    // Fail - "Exception thrown: read access violation."
    //xTaskCreate(t7_DirectParams, "t7_DirectParams", 100, N

    // SURPISE - EXECUTES! - But no delay, instantly continu
    //xTaskCreate(t8_WrongParams_WrongOrder, "t8_WrongParams

    // Fail - "Exception thrown: read access violation."
    //xTaskCreate(t9_WrongParams_WrongType, "t9_WrongParams_


    vTaskStartScheduler();
    return 0;
}

```

## ✓ Key Achievements (Continue)

•Perfect Collaboration between members: each members are equally participating throughout the testing, discussion, decision-making processes.

•In accordance with the work ethic, members are firmly accountable for their roles.

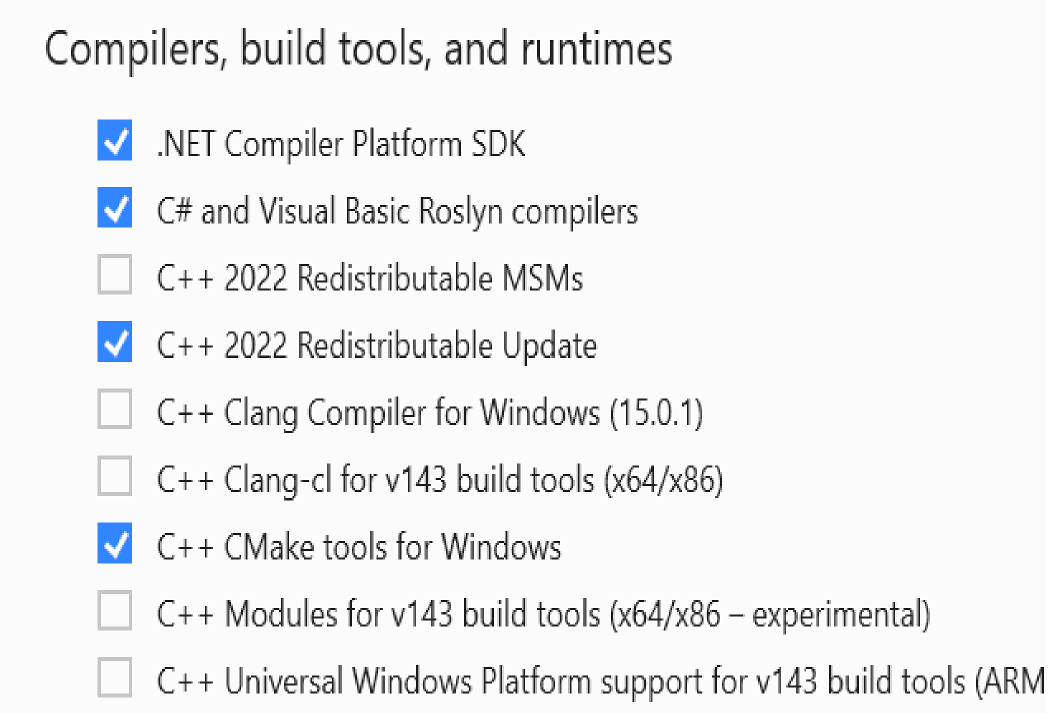


# Project Instruction

Corner Crew

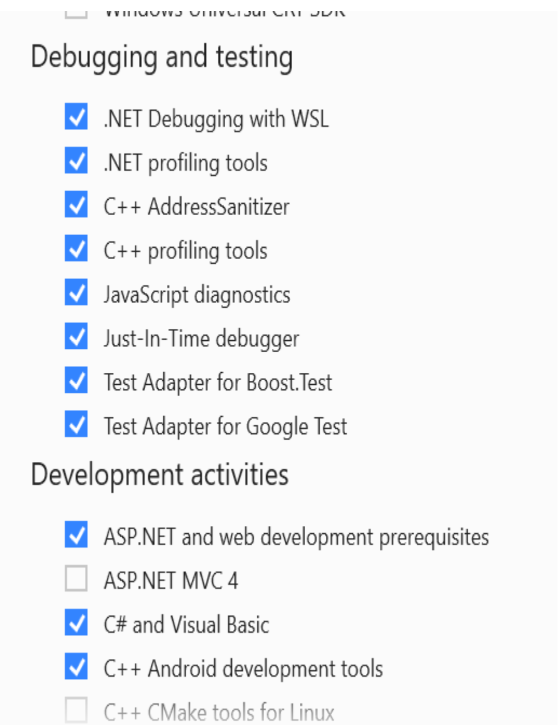
✓ **Project Limitations** - *(Technical: Environment)*

Everyone have different error! - This is due to several reasons : Environment Package, Workload, VS component during the installation, and etc



Compilers, build tools, and runtimes

- ☑ .NET Compiler Platform SDK
- ☑ C# and Visual Basic Roslyn compilers
- □ C++ 2022 Redistributable MSMs
- ☑ C++ 2022 Redistributable Update
- □ C++ Clang Compiler for Windows (15.0.1)
- □ C++ Clang-cl for v143 build tools (x64/x86)
- ☑ C++ CMake tools for Windows
- □ C++ Modules for v143 build tools (x64/x86 – experimental)
- □ C++ Universal Windows Platform support for v143 build tools (ARM



- □ Windows Universal CRT SDK

Debugging and testing

- ☑ .NET Debugging with WSL
- ☑ .NET profiling tools
- ☑ C++ AddressSanitizer
- ☑ C++ profiling tools
- ☑ JavaScript diagnostics
- ☑ Just-In-Time debugger
- ☑ Test Adapter for Boost.Test
- ☑ Test Adapter for Google Test

Development activities

- ☑ ASP.NET and web development prerequisites
- □ ASP.NET MVC 4
- ☑ C# and Visual Basic
- ☑ C++ Android development tools
- □ C++ CMake tools for Linux

# Project Instruction

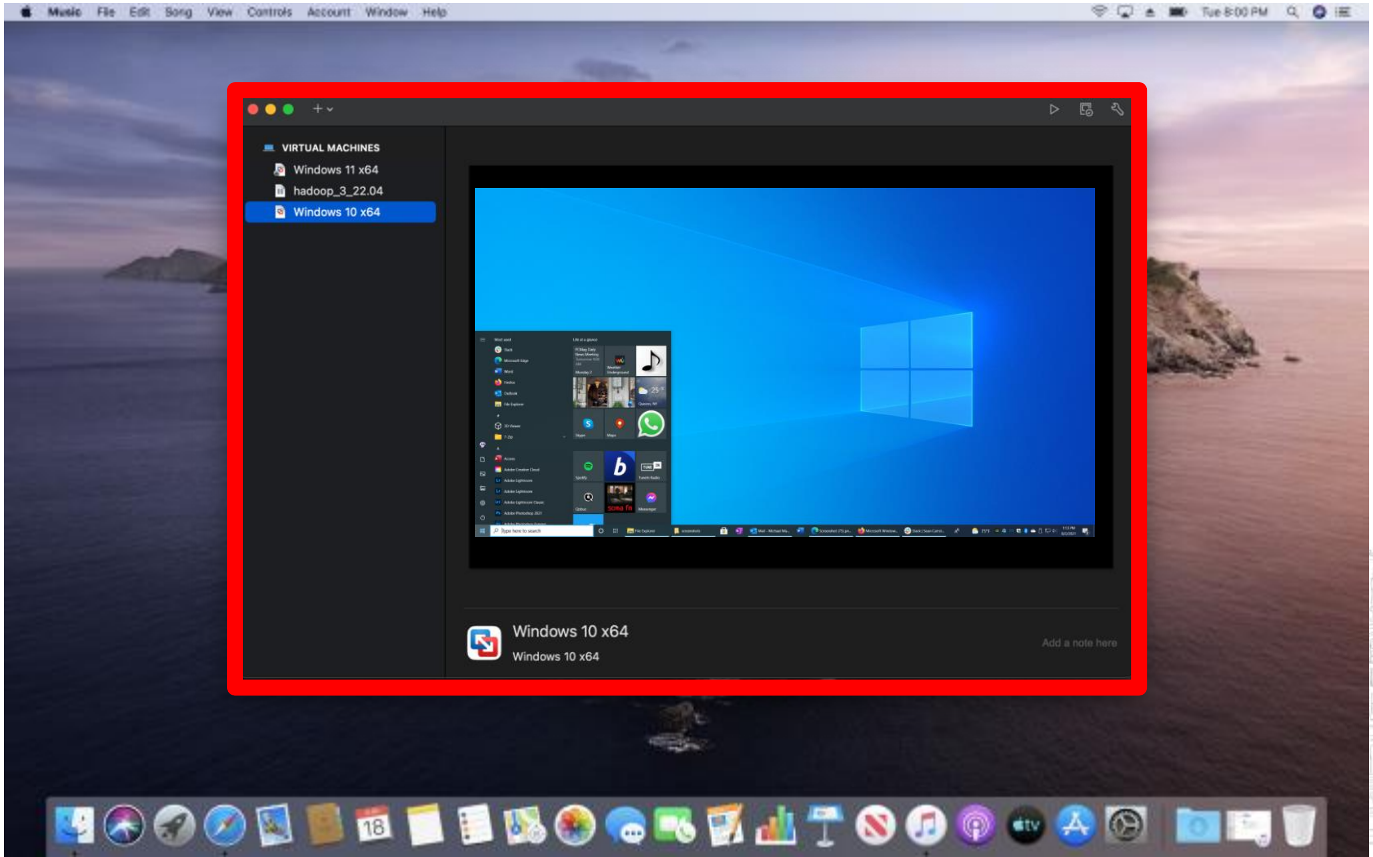
Corner Crew

## ✓ Project Limitations

*(Technical: Environment)*

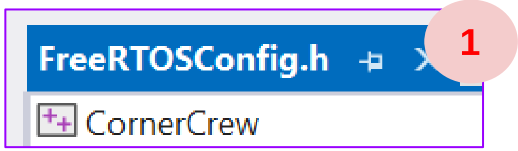
Poor virtual machine support

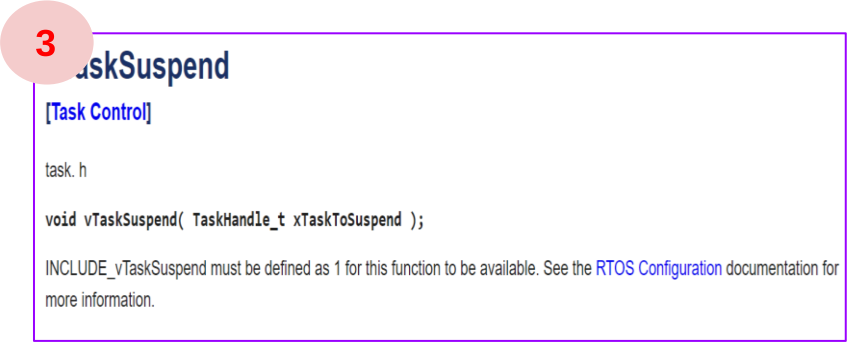
FreeRTOS running in a VM was error prone and slow
So the only choice was to use a Windows installation (Mac problem)



## ✓ **Project Limitations** - *(Technical: Program)*

APIs configuration: some API have to directly configure in the ‘FreeRTOSConfig.h’ file





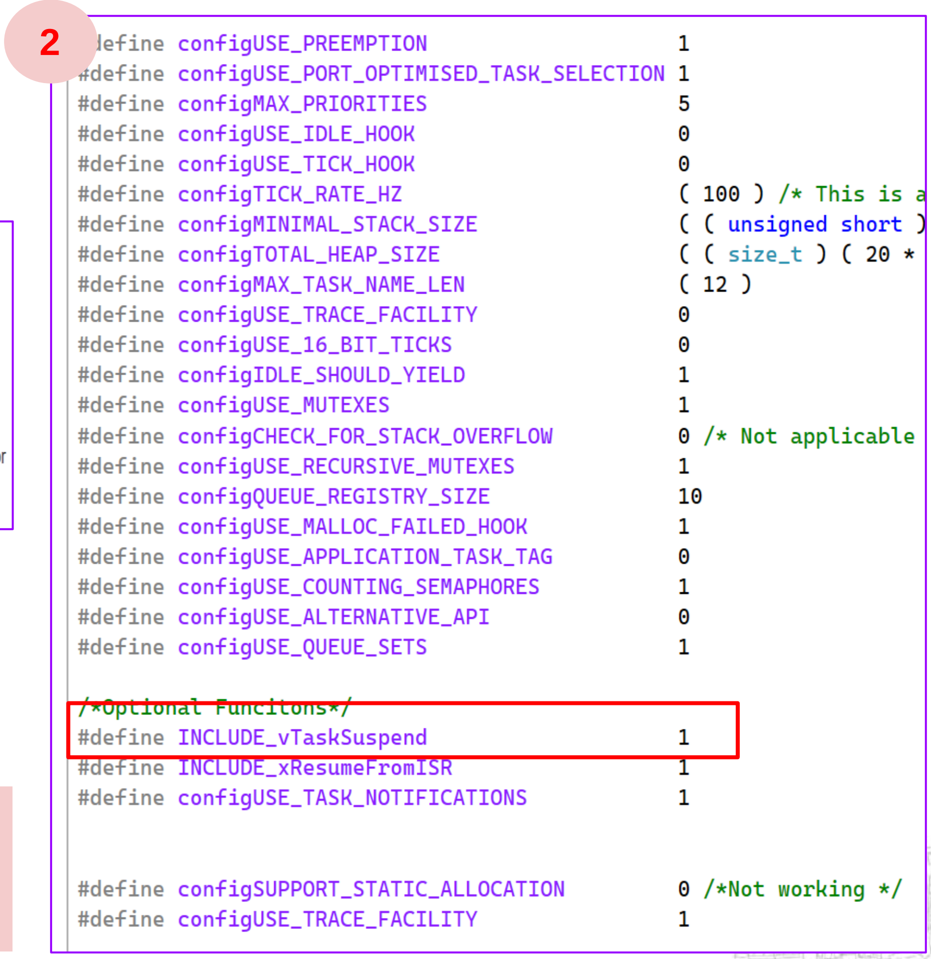
### askSuspend

[Task Control]

task. h

```
void vTaskSuspend( TaskHandle_t xTaskToSuspend );
```

INCLUDE_vTaskSuspend must be defined as 1 for this function to be available. See the RTOS Configuration documentation for more information.



```
#define configUSE_PREEMPTION                    1
#define configUSE_PORT_OPTIMISED_TASK_SELECTION 1
#define configMAX_PRIORITIES                    5
#define configUSE_IDLE_HOOK                     0
#define configUSE_TICK_HOOK                     0
#define configTICK_RATE_HZ                      ( 100 ) /* This is a
#define configMINIMAL_STACK_SIZE                ( ( unsigned short )
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) ( 20 *
#define configMAX_TASK_NAME_LEN                 ( 12 )
#define configUSE_TRACE_FACILITY                0
#define configUSE_16_BIT_TICKS                  0
#define configIDLE_SHOULD_YIELD                 1
#define configUSE_MUTEXES                       1
#define configCHECK_FOR_STACK_OVERFLOW          0 /* Not applicable
#define configUSE_RECURSIVE_MUTEXES             1
#define configQUEUE_REGISTRY_SIZE               10
#define configUSE_MALLOC_FAILED_HOOK            1
#define configUSE_APPLICATION_TASK_TAG          0
#define configUSE_COUNTING_SEMAPHORES           1
#define configUSE_ALTERNATIVE_API               0
#define configUSE_QUEUE_SETS                    1

/*Optional Funcitons*/
#define INCLUDE_vTaskSuspend                    1
#define INCLUDE_xResumeFromISR                  1
#define configUSE_TASK_NOTIFICATIONS            1


#define configSUPPORT_STATIC_ALLOCATION         0 /*Not working */
#define configUSE_TRACE_FACILITY                1
```

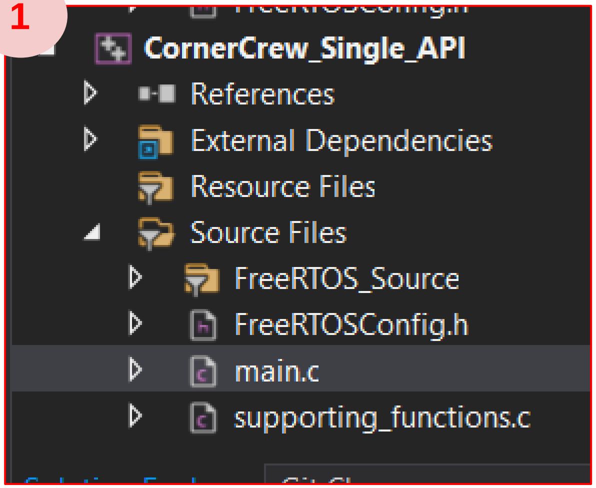
**For example,** to test the vTaskSuspend API, the ‘INCLUDE_vTaskSuspend’ function must define to value ‘1’ to be available.

Drawback: sometimes although the function is configed, the program still isn’t rendering.

## ✓ **Project Limitations** - File structure in VS is independent from File system

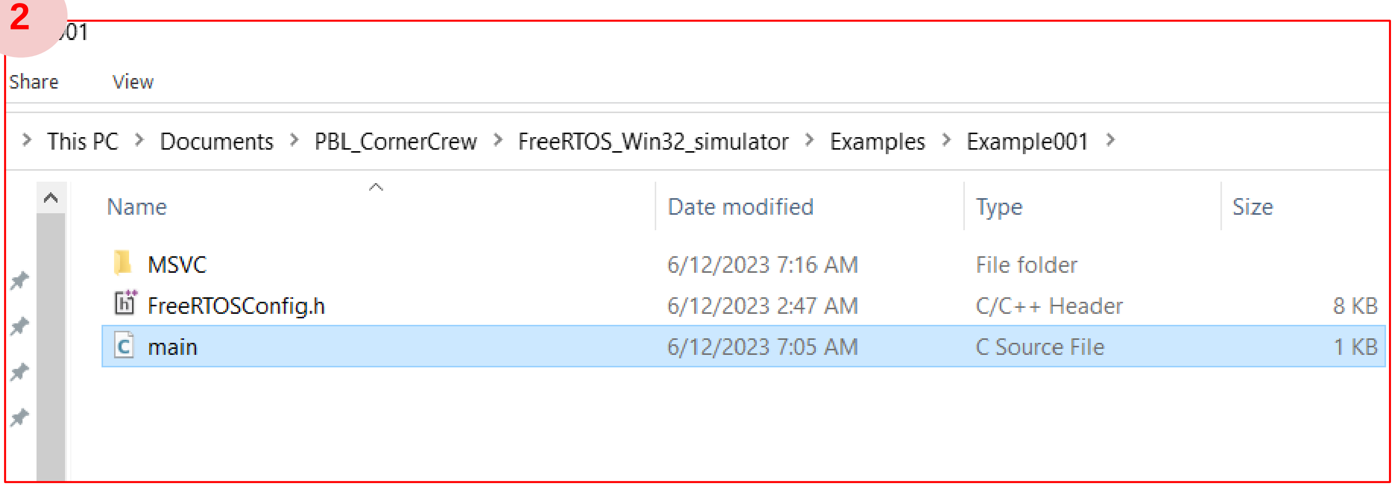
***(Technical: Program)***

1



- CornerCrew_Single_API
  - References
  - External Dependencies
  - Resource Files
  - Source Files
    - FreeRTOS_Source
    - FreeRTOSConfig.h
    - main.c
    - supporting_functions.c

2



This PC › Documents › PBL_CornerCrew › FreeRTOS_Win32_simulator › Examples › Example001 ›

| Name | Date modified | Type | Size |
|---|---|---|---|
| MSVC | 6/12/2023 7:16 AM | File folder | |
| FreeRTOSConfig.h | 6/12/2023 2:47 AM | C/C++ Header | 8 KB |
| main | 6/12/2023 7:05 AM | C Source File | 1 KB |

# Project Instruction

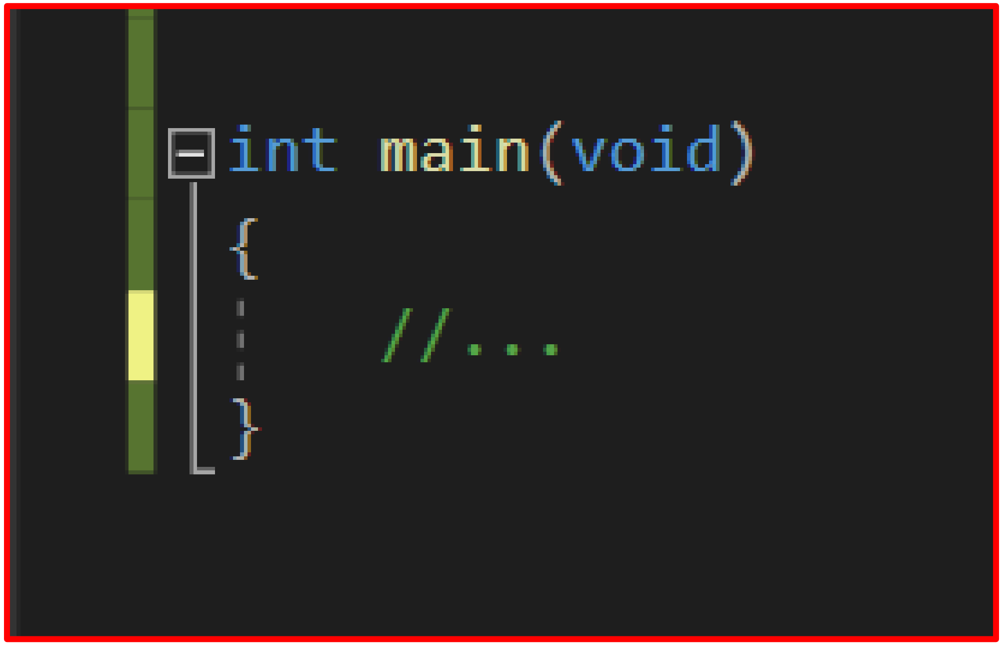
## ✓ Project Limitations

*(Technical: Program)*

Strict 1 name across the whole project. only 1 main()

C and FreeRTOS is very strongly typed.
So it is very strict about the names of variables and functions

Only 1 file is allowed a main() function, and other strict C features

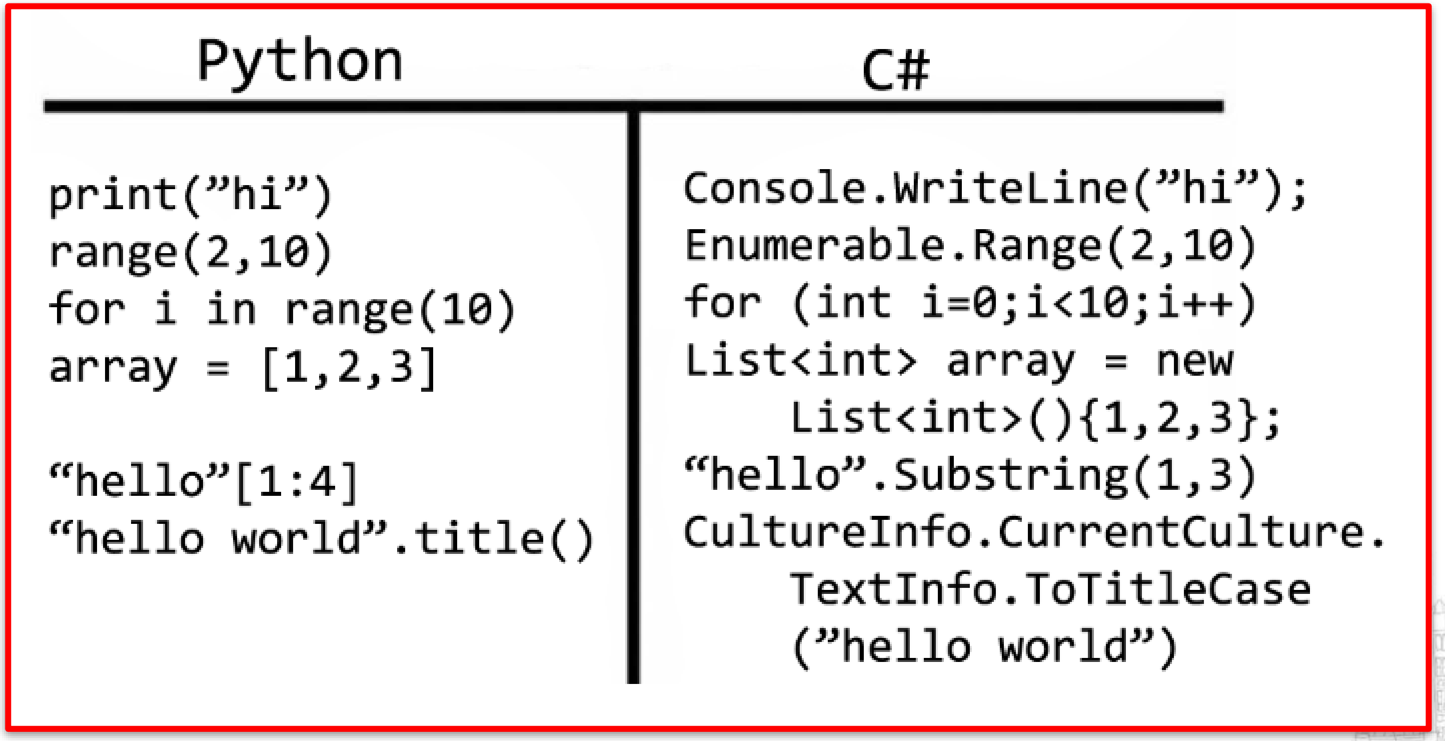


```
int main(void)
{
    //...
}
```

## ✓ Project Limitations - Limitations because of C

***(Technical: Program)***

In Python, it is very easy to loop, split, or any other number of functions
But in C, all these basic things are very challenging
So, consequently, we avoid them.

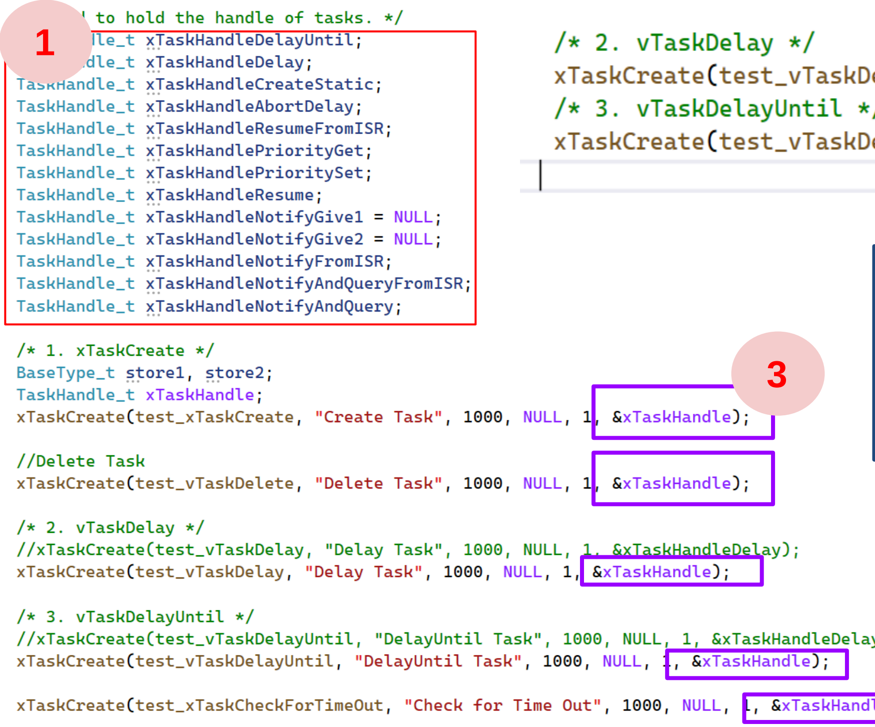


| Python | C# |
|---|---|
| `print(”hi”)` | `Console.WriteLine(”hi”);` |
| `range(2,10)` | `Enumerable.Range(2,10)` |
| `for i in range(10)` | `for (int i=0;i<10;i++)` |
| `array = [1,2,3]` | `List<int> array = new`<br>`List<int>(){1,2,3};` |
| `“hello”[1:4]` | `“hello”.Substring(1,3)` |
| `“hello world”.title()` | `CultureInfo.CurrentCulture.`<br>`TextInfo.ToTitleCase`<br>`(”hello world”)` |

## ✓ Project Limitations

*(Individual)*

- Create too many task handle: create an excessive number of tasks, may exhaust system resources, such as stack space and control structures, to function properly.

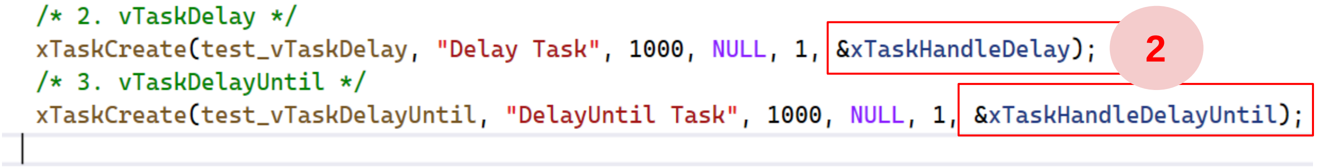


1

```
  to hold the handle of tasks. */
   lle_t xTaskHandleDelayUntil;
   dle_t xTaskHandleDelay;
TaskHandle_t xTaskHandleCreateStatic;
TaskHandle_t xTaskHandleAbortDelay;
TaskHandle_t xTaskHandleResumeFromISR;
TaskHandle_t xTaskHandlePriorityGet;
TaskHandle_t xTaskHandlePrioritySet;
TaskHandle_t xTaskHandleResume;
TaskHandle_t xTaskHandleNotifyGive1 = NULL;
TaskHandle_t xTaskHandleNotifyGive2 = NULL;
TaskHandle_t xTaskHandleNotifyFromISR;
TaskHandle_t xTaskHandleNotifyAndQueryFromISR;
TaskHandle_t xTaskHandleNotifyAndQuery;
```

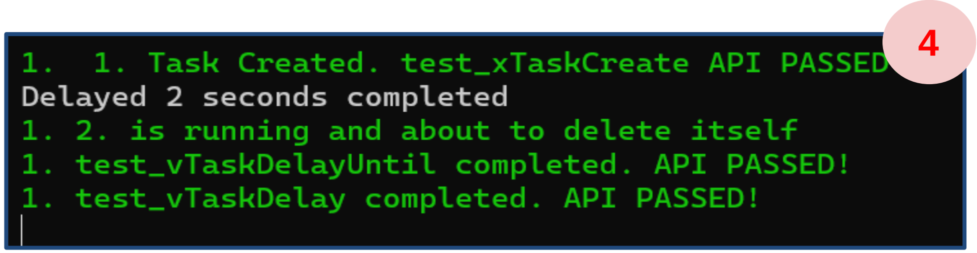
3

```
/* 1. xTaskCreate */
BaseType_t store1, store2;
TaskHandle_t xTaskHandle;
xTaskCreate(test_xTaskCreate, "Create Task", 1000, NULL, 1, &xTaskHandle);

//Delete Task
xTaskCreate(test_vTaskDelete, "Delete Task", 1000, NULL, 1, &xTaskHandle);

/* 2. vTaskDelay */
//xTaskCreate(test_vTaskDelay, "Delay Task", 1000, NULL, 1, &xTaskHandleDelay);
xTaskCreate(test_vTaskDelay, "Delay Task", 1000, NULL, 1, &xTaskHandle);

/* 3. vTaskDelayUntil */
//xTaskCreate(test_vTaskDelayUntil, "DelayUntil Task", 1000, NULL, 1, &xTaskHandleDelay
xTaskCreate(test_vTaskDelayUntil, "DelayUntil Task", 1000, NULL, 1, &xTaskHandle);

xTaskCreate(test_xTaskCheckForTimeOut, "Check for Time Out", 1000, NULL, 1, &xTaskHandle);
```



2

```
/* 2. vTaskDelay */
xTaskCreate(test_vTaskDelay, "Delay Task", 1000, NULL, 1, &xTaskHandleDelay);
/* 3. vTaskDelayUntil */
xTaskCreate(test_vTaskDelayUntil, "DelayUntil Task", 1000, NULL, 1, &xTaskHandleDelayUntil);
```



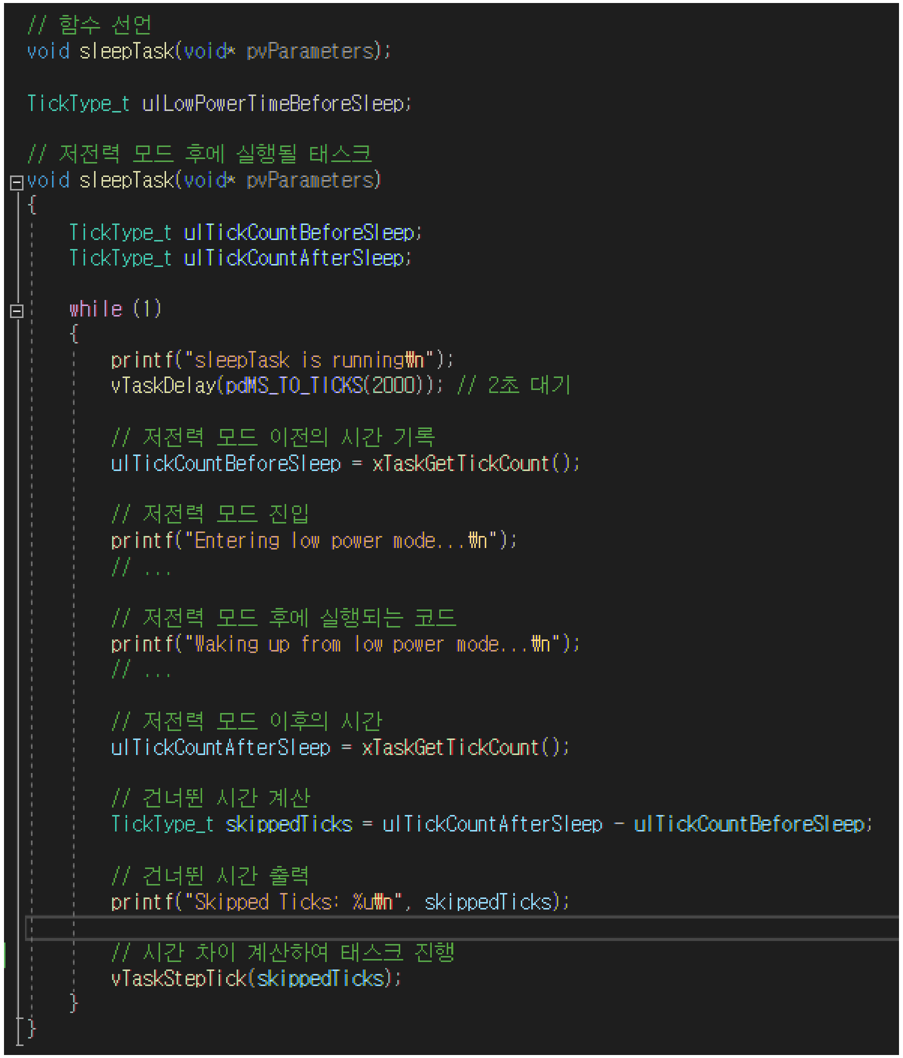
4

```
1.  1. Task Created. test_xTaskCreate API PASSED
Delayed 2 seconds completed
1. 2. is running and about to delete itself
1. test_vTaskDelayUntil completed. API PASSED!
1. test_vTaskDelay completed. API PASSED!
```

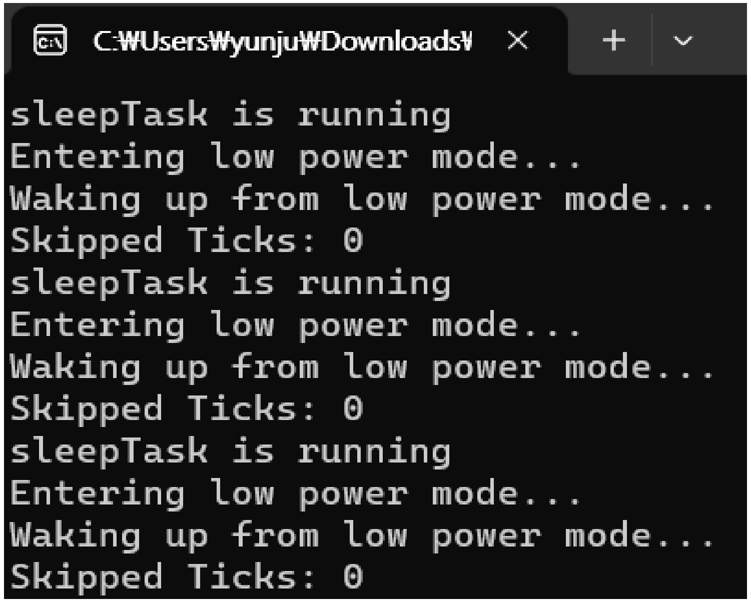
To determine the optimal number of tasks necessary to accomplish project goals without overburdening the resources of the system. We tried to re-used the same declared task handle.

## ✓ Project Limitations

*(Individual)*



```
// 함수 선언
void sleepTask(void* pvParameters);

TickType_t ulLowPowerTimeBeforeSleep;

// 저전력 모드 후에 실행될 태스크
void sleepTask(void* pvParameters)
{
    TickType_t ulTickCountBeforeSleep;
    TickType_t ulTickCountAfterSleep;

    while (1)
    {
        printf("sleepTask is running₩n");
        vTaskDelay(pdMS_TO_TICKS(2000)); // 2초 대기

        // 저전력 모드 이전의 시간 기록
        ulTickCountBeforeSleep = xTaskGetTickCount();

        // 저전력 모드 진입
        printf("Entering low power mode...₩n");
        // ...

        // 저전력 모드 후에 실행되는 코드
        printf("Waking up from low power mode...₩n");
        // ...

        // 저전력 모드 이후의 시간
        ulTickCountAfterSleep = xTaskGetTickCount();

        // 건너뛴 시간 계산
        TickType_t skippedTicks = ulTickCountAfterSleep - ulTickCountBeforeSleep;

        // 건너뛴 시간 출력
        printf("Skipped Ticks: %u₩n", skippedTicks);

        // 시간 차이 계산하여 태스크 진행
        vTaskStepTick(skippedTicks);
    }
}
```



```
sleepTask is running
Entering low power mode...
Waking up from low power mode...
Skipped Ticks: 0
sleepTask is running
Entering low power mode...
Waking up from low power mode...
Skipped Ticks: 0
sleepTask is running
Entering low power mode...
Waking up from low power mode...
Skipped Ticks: 0
```

'vTaskStepTick()' is a function in FreeRTOS that moves the system tick forward by a specified amount of time. It is typically used in low-power mode to skip a system tick and update the system state by skipping some time.

However, if you look at the terminal, it doesn't run because nothing is skipped. I have not been able to resolve this error.

# Project Instruction

Corner Crew

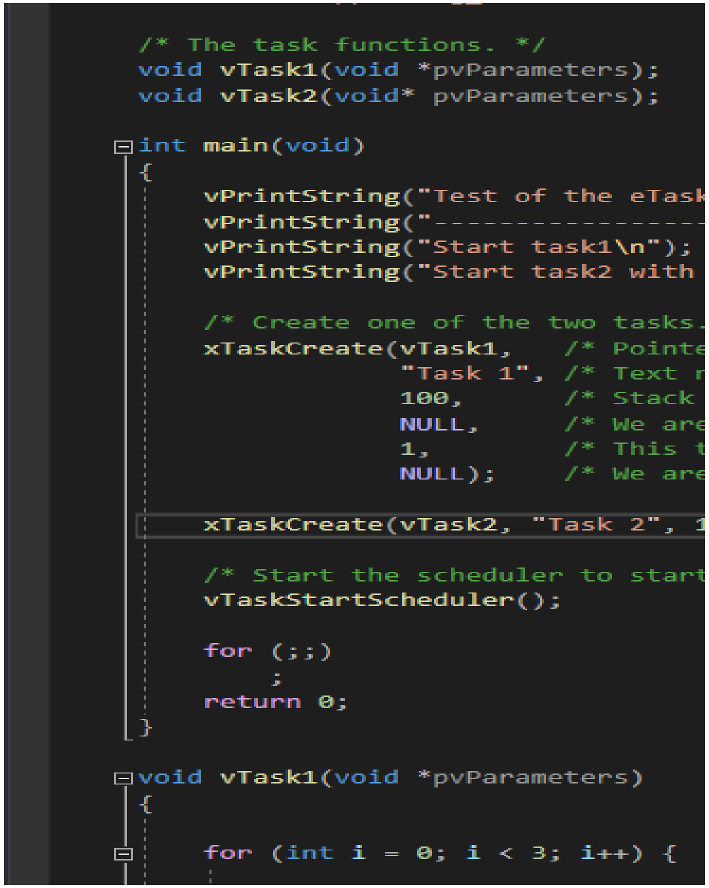
## ✓ Project Limitations

***(Individual)***

- Task can be very simple

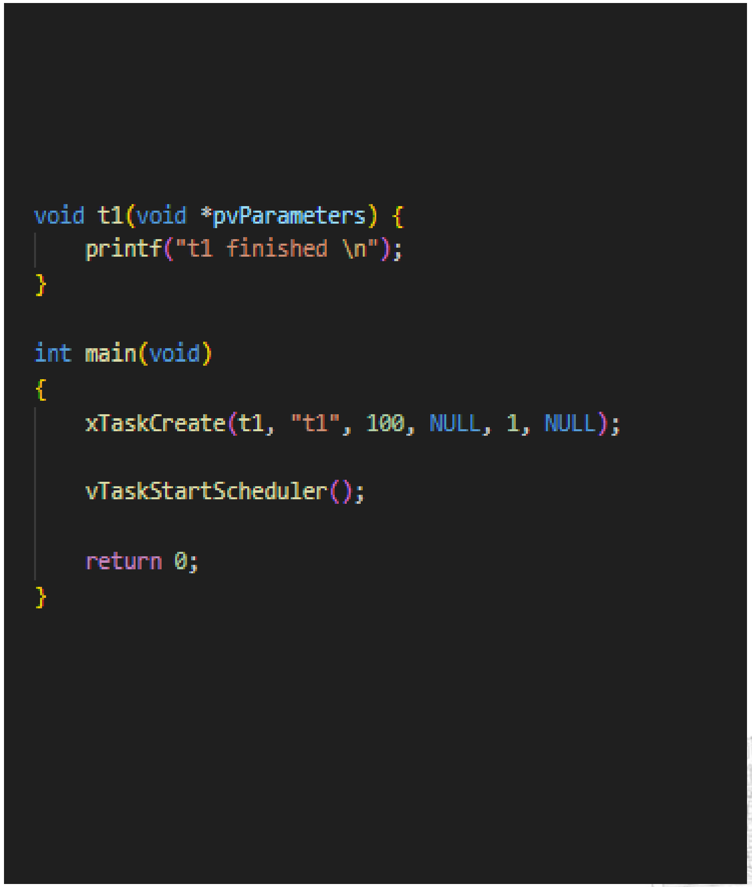
  After many tasks, we discovered that very little code is required to actually run a task

**Long**



```c
    /* The task functions. */
    void vTask1(void *pvParameters);
    void vTask2(void* pvParameters);

int main(void)
{
        vPrintString("Test of the eTask
        vPrintString("------------------
        vPrintString("Start task1\n");
        vPrintString("Start task2 with

        /* Create one of the two tasks.
        xTaskCreate(vTask1,    /* Pointe
                    "Task 1", /* Text r
                    100,       /* Stack
                    NULL,      /* We are
                    1,         /* This t
                    NULL);     /* We are

        xTaskCreate(vTask2, "Task 2", 1

        /* Start the scheduler to start
        vTaskStartScheduler();

        for (;;)
            ;
        return 0;
}

void vTask1(void *pvParameters)
{

        for (int i = 0; i < 3; i++) {
```

**Short**



```c
void t1(void *pvParameters) {
    printf("t1 finished \n");
}

int main(void)
{
    xTaskCreate(t1, "t1", 100, NULL, 1, NULL);

    vTaskStartScheduler();

    return 0;
}
```

# Summary of Project

## ✓ Project Purpose

- Enhance FreeRTOS API understanding: after actively testing almost all of the FreeRTOS APIs we have gained practical experience and familiarity with its functionality, usage, and behavior of the APIs, particularly understood how C language work better.

- Improve code stability and quality: through code review, testing, and debugging, we will gain a better understanding of implementing a reliable and robust software that may use FreeRTOS APIs.

- Share Knowledge and Collaboration: this project encourages knowledge sharing and actively engaging in discussion.

# Summary of Project

## ✓ Project Background

- The project is undertaken to thoroughly test and implement the 55 FreeRTOS APIs.
- It is one of a popular real-time operating system used in embedded systems and IoT applications.
- Amazon recently developed FreeRTOS, including Space-X's manned spacecraft.
- However, due to the urgent development timeline, certain issues were identified in the Task Management & Scheduling functionality of FreeRTOS.

# Summary of Project

## ✓ Project Scope

- Task Management & Scheduling Testing: focus on testing and addressing the identified issues(if there is any).

- Validation and Verification: reviewing the documentation and verify if the documented API produce an expected result.

- Testing and Validation Methodologies: define our testing strategy.

# Summary of Project

## ✓ Milestone

| Milestones | Description |
|---|---|
| Milestone 1: Familiarize with test environment | - Set up testing environment (MS Visual Studio)<br>- Team member familiarize the APIs by reading the documentation |
| Milestone 2: Implement base file | - Create new blinky file to test the API |
| Milestone 3: Initial API testing | - Test 40 APIs<br>- Validate tested APIs with provided documentation |
| Milestone 4: Report and Debug | - Member report encountered issue<br>- Resolve the encountered issue together |

# Summary of Project

## ✓ Milestone

| Milestones | Description |
|---|---|
| Milestone 5: Generated A Quality Test Report | Documented tested APIs' behaviour and usage. |

# Summary of Project

## ✓ Methodology for dividing API's evenly among members

Corresponding to the previous week question regarding distribute task management to members for a better productive result

- Anyone can choose any uncompleted API.

- To ensure collaboration and knowledge sharing, team members must share the code for their chosen API with others.

- Sharing the code allows other team members to reference it in case they also need to use that API.

- This promotes an efficient and collaborative workflow, enabling team members to support each other and avoid duplicating efforts.

# Project Execution

## ✓ Decision behind the chosen methods

- Unit testing is a standard practice in all areas of software development

- Therefore it is a common method
    - That means other developers can quickly understand the layout of the tests
    - And what to expect from each function
        - E.G: this API will output something, or nothing to prove it works
        - Or, stop execution and output an error

# Project Execution

## ✓ Test Execution Plan

Our execution plan for testing all APIs are:

- Pick a single API
- Write a function that uses the API
- Record the result (screenshot)
- Add any extra details (config, issues)
- Move and store the code, move onto next API

# Project Execution

## ✓ How to solve the design problem

- FreeRTOS uses C
- This was the biggest problem
- It made working with the code challenging
- There are many strict rules in C, that made it difficult to manage the many APIs

- Visual Studio was another problem
- Moving files around is hard, adding and deleting files is slow and inconvenient

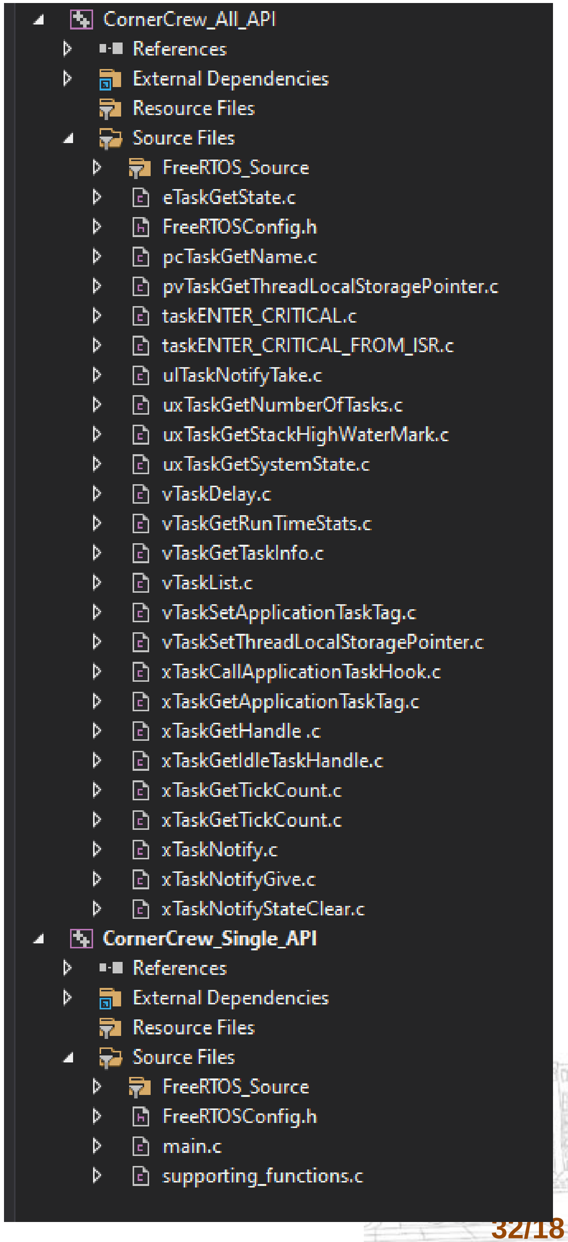
# Project Execution

✓ **How the purpose of the project is accomplished in the final design & execution?**

- We successfully implemented 40 APIs
- And confirmed they worked as expected in basic use cases
- If we had more time, and skill(in C and OS), we could have made extensive test cases for each API
- But at minimum we tested the basic usage of 40 APIs

# Description of Final Design

- Our final design was to split the code into 2 projects
- This was done because C is very strict
- So if you want to test an API, you must copy the code from _All_API over to _Single_API
- This was the most organised method



```
◢ CornerCrew_All_API
  ▷ References
  ▷ External Dependencies
    Resource Files
  ◢ Source Files
    ▷ FreeRTOS_Source
    ▷ eTaskGetState.c
    ▷ FreeRTOSConfig.h
    ▷ pcTaskGetName.c
    ▷ pvTaskGetThreadLocalStoragePointer.c
    ▷ taskENTER_CRITICAL.c
    ▷ taskENTER_CRITICAL_FROM_ISR.c
    ▷ ulTaskNotifyTake.c
    ▷ uxTaskGetNumberOfTasks.c
    ▷ uxTaskGetStackHighWaterMark.c
    ▷ uxTaskGetSystemState.c
    ▷ vTaskDelay.c
    ▷ vTaskGetRunTimeStats.c
    ▷ vTaskGetTaskInfo.c
    ▷ vTaskList.c
    ▷ vTaskSetApplicationTaskTag.c
    ▷ vTaskSetThreadLocalStoragePointer.c
    ▷ xTaskCallApplicationTaskHook.c
    ▷ xTaskGetApplicationTaskTag.c
    ▷ xTaskGetHandle .c
    ▷ xTaskGetIdleTaskHandle.c
    ▷ xTaskGetTickCount.c
    ▷ xTaskGetTickCount.c
    ▷ xTaskNotify.c
    ▷ xTaskNotifyGive.c
    ▷ xTaskNotifyStateClear.c
◢ CornerCrew_Single_API
  ▷ References
  ▷ External Dependencies
    Resource Files
  ◢ Source Files
    ▷ FreeRTOS_Source
    ▷ FreeRTOSConfig.h
    ▷ main.c
    ▷ supporting_functions.c
```

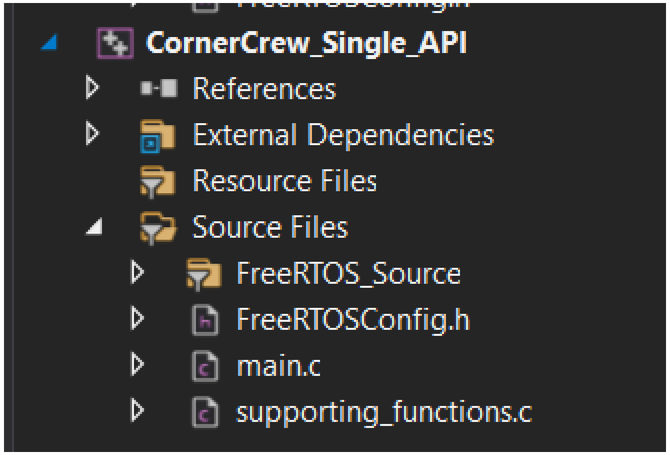
# Conclusion & Recommendation

## ✓ Conclusion and Suggestions

- There were many challenges and learning experiences during this project
- Working with C was an entirely new experience
- You must understand how Operating systems work if you want to use FreeRTOS

- Suggestion - Maybe Quality is better than Quantity
  - Perhaps it is better to test fewer APIs
  - maybe 20
  - but test each API 100%
  - with at least 4 unique tests per API

# LIVE API DEMO



- CornerCrew_Single_API
  - References
  - External Dependencies
  - Resource Files
  - Source Files
    - FreeRTOS_Source
    - FreeRTOSConfig.h
    - main.c
    - supporting_functions.c

# Conclusion & Recommendation

## ✓ Expected outcome if given an additional two weeks

- Finish testing all API’s
- Test each API with multiple test cases
- Find a better way to store and manage test functions
    - Multiple files
    - Multtple main() functions

Thank You ~

If you might have any questions.
Don't ask! 🤪