Main

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "uart_driver.h"
#include "delay.h"
#include "music.h"
#include "memconsole.h"
#include "period.h"
#include "music.h"
#include "waveforms.h"
#include "console.h"
#include "multitask.h"
#include "LED.h"

#define F_CPU 16000000UL
#define MOTION 50

void krl() {
	while(1==1) {
		int light = 1;

		for(int i = 0; i < 10; i++) {
			int shifted = light << i;
			LED_PrintNum(shifted);
			delay_ms(MOTION);
		}

		light = 1 << 9;

		for(int i = 0; i < 10; i++) {
			int shifted = light >> i;
			LED_PrintNum(shifted);
			delay_ms(MOTION);
		}
	}
}

int main(void) {
	init_usart2(115200,F_CPU);

	delay_Init();

	music_Init();

	//period_Init();

	wave_Init();

	LED_Init();

	init_tasker(2, 10);
	init_task(1, 1000, console, 5);

	stk_Init();

	krl();
	exit(EXIT_SUCCESS);
	return 0;
}
```

Multitasking Header

```
#ifndef MULTITASK_IS_ALIVE
#define MULTITASK_IS_ALIVE 1

#include <stdint.h>

typedef enum{PAUSED,ACTIVE} task_state;

#define SCB_ICSR (uint32_t*) 0xE000ED04
#define PENDSVSET 28

typedef struct {
	uint32_t* stack_pointer;
	task_state state;
	uint32_t ticks_starting;
	uint32_t ticks_remaining;
} task;

void tasker_tick();
void init_tasker(uint32_t total_tasks, uint32_t main_ticks);
void init_task(uint32_t task_num, uint32_t stack_size, void(*entry_point)(void), uint32_t ticks);


void PendSV_Handler(void) __attribute__((naked));
void stk_Init();
void SysTick_Handler();

#endif
```

Console Header

```
#ifndef CONSOLE_IS_ALIVE
#define CONSOLE_IS_ALIVE 1

void console();

#endif
```

Console

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "uart_driver.h"
#include "delay.h"
#include "music.h"
#include "memconsole.h"
#include "period.h"
#include "music.h"
#include "waveforms.h"

// Rip and tear until it is done
#define doomTempo 1500000 // Technically this track should be 240bpm but this sounds right
static note atDoomsGate[] = {
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {C, 5, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {B, 4, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {A, 4, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {Fs, 4, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {G, 4, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {A, 4, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {C, 5, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {B, 4, doomTempo>>3},
             {0, 0, doomTempo>>5},
             {F, 2, doomTempo>>3},
```

```
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{G, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{Fs, 4, (doomTempo>>2)+(doomTempo>>3)},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{C, 5, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{B, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{A, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{Fs, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{G, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{A, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{C, 5, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{B, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{G, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
```

```
            {F, 2, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {F, 2, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {Fs, 4, (doomTempo>>2)+(doomTempo>>3)},
            {0, 0, 0}
};

// "130bpm"
#define zeldaTempo 800000       // This totally isn't 130bpm but it sounds right
#define betweenNotes 46000
static note zelda[] = {
            {A, 4, zeldaTempo>>1},
            {0, 1, zeldaTempo>>2},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>2},
            {0, 0, betweenNotes},
            {B, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>2},
            {0, 1, zeldaTempo>>2},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>2},
            {0, 0, betweenNotes},
            {B, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>2},
            {0, 1, zeldaTempo>>2},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {B, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {E, 4, zeldaTempo>>4},
            {0, 0, betweenNotes},
            {E, 4, zeldaTempo>>4},
            {0, 0, betweenNotes},
            {E, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {E, 4, zeldaTempo>>4},
            {0, 0, betweenNotes},
            {E, 4, zeldaTempo>>4},
```

```
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>4},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>4},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{A, 4, zeldaTempo>>2},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>2},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{A, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{A, 4, zeldaTempo>>4},
{0, 0, betweenNotes},
{B, 4, zeldaTempo>>4},
{0, 0, betweenNotes},
{C, 5, zeldaTempo>>4},
{0, 0, betweenNotes},
{D, 5, zeldaTempo>>4},
{0, 0, betweenNotes},
{E, 5, zeldaTempo>>1},
{0, 1, zeldaTempo>>3},
{D, 5, zeldaTempo>>3},
{0, 0, betweenNotes},
{E, 5, zeldaTempo>>3},
{0, 0, betweenNotes},
{F, 5, zeldaTempo>>3},
{0, 0, betweenNotes},
{G, 5, zeldaTempo>>3},
{0, 0, betweenNotes},
{B, 6, zeldaTempo>>5},
{0, 0, betweenNotes},
{D, 6, zeldaTempo>>3},
{0, 0, betweenNotes},
{B, 6, zeldaTempo>>3},
{0, 0, betweenNotes},
{G, 5, zeldaTempo>>3},
{0, 0, betweenNotes},
{F, 5, zeldaTempo>>1},
{0, 0, betweenNotes},
{G, 5, zeldaTempo>>4},
{0, 1, zeldaTempo>>4},
{F, 5, zeldaTempo>>4},
{0, 0, betweenNotes},
{E, 5, zeldaTempo>>1},
{0, 0, betweenNotes},
{E, 5, zeldaTempo>>2},
{0, 0, betweenNotes},
{D, 5, zeldaTempo>>3},
{0, 0, betweenNotes},
{D, 5, zeldaTempo>>4},
{0, 0, betweenNotes},
{E, 5, zeldaTempo>>4},
{0, 0, betweenNotes},
```

```
            {F, 5, zeldaTempo>>1},
            {0, 0, betweenNotes},
            {E, 5, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {D, 5, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {D, 5, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {C, 5, zeldaTempo>>4},
            {0, 0, betweenNotes},
            {D, 5, zeldaTempo>>4},
            {0, 0, betweenNotes},
            {E, 5, zeldaTempo>>1},
            {0, 0, 0}
};

void console() {
      while(1==1) {

      // Blank string for input
      char input[30] = "";

      // Address to interact with
      uint32_t* address = 0;

      // Command variable
      int command = -1;

      uint32_t frequency = 0;
      uint32_t samples = 0;
      uint16_t* waveform;

      // Last argument, either length to read or value to write
      uint32_t argument = 0;

      // Welcome message
      printf("Evan's Memory Management Console\n\r");
      printf("Type \'?\' for help\n\r");

      // Infinite loop for program
      while(1==1) {
            // Prompt
            printf("> ");
            fgets(input, 29, stdin);

            // First token, determines command
            char* strnCommand = strtok(input, " ");

            // Second token
            char* arg1 = strtok(NULL, " ");

            // Third token
            char* arg2 = strtok(NULL, " ");

            // If there is an extracted command
            if(strnCommand != NULL) {
                  // Attempt to parse the command
                  command = parseCommand(strnCommand);

                  // Switch case for reported commands
                  switch (command) {
```

```
// Help command
case 0:
	help();
	break;

// Dump memory command
case 1:
	// Attempt to parse address
	if(arg1 != NULL) {
		address = parseAddress(arg1);
	}

	// Attempt to parse second argument
	if(arg2 != NULL) {
		argument = parseArgument(arg2);
	}

	if(arg1 != NULL) {
		if(arg2 == NULL) {
			memdmpDefault((uint8_t*)address);
		} else {
			memdmp((uint8_t*)address, argument);
		}
	} else {
		printf("No address provided\n\r");
	}
	break;

// Read word command
case 2:
	// Attempt to parse address
	if(arg1 != NULL) {
		address = parseAddress(arg1);
	}

	// Attempt to parse second argument
	if(arg2 != NULL) {
		argument = parseArgument(arg2);
	}

	if(arg1 != NULL) {
		memwrd(address);
	} else {
		printf("No address provided\n\r");
	}
	break;

// Write word command
case 3:
	// Attempt to parse address
	if(arg1 != NULL) {
		address = parseAddress(arg1);
	}

	// Attempt to parse second argument
	if(arg2 != NULL) {
		argument = parseArgument(arg2);
	}
```

```
		if(arg1 != NULL) {
			if(arg2 != NULL) {
				wmemwrd(address, argument);
			} else {
				printf("No value to write provided\n\r");
			}
		} else {
			printf("No address provided\n\r");
		}
		break;

	// Music command
	case 4:
		// Determine song to be played
		if(strcmp(arg1, "doom") == 0 || strcmp(arg1, "doom\n") == 0) {

			// Play background/foreground accordingly
			if(strcmp(arg2, "background\n") == 0) {
				music_Background(atDoomsGate);
			} else {
				music_Play(atDoomsGate);
			}

		} else if(strcmp(arg1, "zelda") == 0 || strcmp(arg1, "zelda\n") == 0) {

			// Play background/foreground accordingly
			if(strcmp(arg2, "background\n") == 0) {
				music_Background(zelda);
			} else {
				music_Play(zelda);
			}

		} else {
			printf("Invalid song\n");
		}
		break;

	// Frequency Measurement
	case 5:
		if(arg1 != NULL) {
			if(strcmp(arg1, "frequency\n") == 0) {
				printf("\nMeasuring frequency...\n\n");
				double average = period_Measure();
				printf("Measured frequency was %.2f Hz\n", average);
				} else {
					printf("Invalid measurement\n");
				}
			} else {
				printf("Measurement type required\n");
			}
			break;
```

```
// Sine wave command
case 6:
	// Parse Frequency
	if(arg1 != NULL) {
		frequency = parseArgument(arg1);
	} else {
		printf("No frequency provided\n");
	}

	// Parse Samples
	if(arg2 != NULL) {
		samples = parseArgument(arg2);
	} else {
		printf("No number of samples provided\n");
	}

	// Execute Command
	if(arg1 != NULL && arg2 != NULL) {
		waveform = sineWave(samples);
		wave_Start(waveform, frequency, samples);
	}
	break;

// Sawtooth wave command
case 7:
	// Parse Frequency
	if(arg1 != NULL) {
		frequency = parseArgument(arg1);
	} else {
		printf("No frequency provided\n");
	}

	// Parse Samples
	if(arg2 != NULL) {
		samples = parseArgument(arg2);
	} else {
		printf("No number of samples provided\n");
	}

	// Execute Command
	if(arg1 != NULL && arg2 != NULL) {
		waveform = sawtoothWave(samples);
		wave_Start(waveform, frequency, samples);
	}
	break;
```

```
                // Triangle wave command
                case 8:
                        // Parse Frequency
                        if(arg1 != NULL) {
                                frequency = parseArgument(arg1);
                        } else {
                                printf("No frequency provided\n");
                        }

                        // Parse Samples
                        if(arg2 != NULL) {
                                samples = parseArgument(arg2);
                        } else {
                                printf("No number of samples provided\n");
                        }

                        // Execute Command
                        if(arg1 != NULL && arg2 != NULL) {
                                waveform = triWave(samples);
                                wave_Start(waveform, frequency, samples);
                        }
                        break;

                // Stop waveform command
                case 9:
                        wave_Stop();

                        // Free the malloc
                        free((void*) waveform);
                        break;

                default:
                        printf("Invalid command\n\r");
                }

        } else {
                printf("No input\n\r");
        }

        // fgets again because it will read the newline from previous entry
        fgets(input, 29, stdin);

        // Clear the input string
        memset(input, 0, strlen(input));

    }

    }
    return;
}
```

Multitasking code

```c
// I need the header for the task and task_state typedefs
#include "multitask.h"
#include "registers_new.h"
#include <stdint.h>
#include <stdlib.h>

static task* tasks;
static uint32_t num_tasks;
static uint32_t current_task;
static uint32_t next_task;

void stk_Init() {
	// Systick struct
	volatile SYSTICK* STK = (SYSTICK*)STK_BASE;

	// Enable Systick interrupts
	STK->CTRL |= STK_INT;

	// Set to count according to internal clock
	STK->CTRL |= STK_CLK;

	// Set reload value to 1ms at 16MHz = 16,000
	STK->LOAD = 16000-1;

	STK->CTRL |= STK_EN;
}

void init_tasker(uint32_t total_tasks, uint32_t main_ticks) {
	num_tasks = total_tasks;

	tasks = calloc(total_tasks,sizeof(task));

	tasks[0].state = ACTIVE;
	tasks[0].ticks_starting = main_ticks;
	tasks[0].ticks_remaining = main_ticks;

	current_task = 0;
}

void init_task(uint32_t task_num, uint32_t stacksize, void(*entry_point)(void), uint32_t ticks) {
	tasks[task_num].stack_pointer = (uint32_t*)malloc(stacksize*sizeof(uint32_t));

	tasks[task_num].stack_pointer += stacksize;
```

```
	*(--tasks[task_num].stack_pointer) = 0x01000000;
	*(--tasks[task_num].stack_pointer) = (uint32_t)entry_point;
	*(--tasks[task_num].stack_pointer) = 0xFFFFFFFF;
	*(--tasks[task_num].stack_pointer) = 0x0;
	*(--tasks[task_num].stack_pointer) = 0x0;
	*(--tasks[task_num].stack_pointer) = 0x0;
	*(--tasks[task_num].stack_pointer) = 0x0;
	*(--tasks[task_num].stack_pointer) = 0x0;
	*(--tasks[task_num].stack_pointer) = 0xFFFFFFF9;
	*(--tasks[task_num].stack_pointer) = 0x0;
	*(--tasks[task_num].stack_pointer) = 0x0;
	*(--tasks[task_num].stack_pointer) = 0x0;
	*(--tasks[task_num].stack_pointer) = 0x0;
	*(--tasks[task_num].stack_pointer) = 0x0;
	*(--tasks[task_num].stack_pointer) = 0x0;
	*(--tasks[task_num].stack_pointer) = 0x0;
	*(--tasks[task_num].stack_pointer) = 0x0;

	tasks[task_num].state = ACTIVE;
	tasks[task_num].ticks_starting = ticks;
	tasks[task_num].ticks_remaining = 0;
}

void tasker_tick() {
	tasks[current_task].ticks_remaining--;

	if(tasks[current_task].ticks_remaining == 0) {
		uint32_t i = 1;
		while(tasks[(next_task=(current_task+i)%num_tasks)].state!=ACTIVE) {
			i++;
		}

		tasks[next_task].ticks_remaining = tasks[next_task].ticks_starting;

		*SCB_ICSR |= 1<<PENDSVSET;
	}
}

// ISR to swap tasks
void PendSV_Handler(void) {
	register uint32_t* stack_pointer asm("sp");
	asm volatile ("push {R4-R11,LR}");

	tasks[current_task].stack_pointer = stack_pointer;
	current_task = next_task;
	stack_pointer = tasks[current_task].stack_pointer;

	asm volatile("pop {R4-R11,LR}\n\t" "BX LR");
}



void SysTick_Handler() {
	volatile SYSTICK* STK = (SYSTICK*)STK_BASE;
	STK->VAL = 0;
	tasker_tick();
}
```

My Experience

This lab was kind of a pain to get working properly for me. I had to delete and remake the multitasking source code like 4 different times, and ultimately the problem was an issue with hardware, or at least that's what it seemed like. I had to borrow a classmate's board to see if the code worked, and sure enough it did. On his board.

When I reconnected my board and programmed the code one more time, it decided to work properly. Besides that, it was just splitting the different tasks into different subroutines for multitasking.