```c
/*
 * CE2812 - 021
 * Winter 2016
 * Lab 5 - Console Application
 * Name: Yahui Liang
 * Created: 01/16/2017
 */

/* All header files */
#include <stdio.h>
#include <inttypes.h>
#include <string.h>
#include <stdlib.h>
#include <limits.h>
#include "uart_driver.h"

// a simple menu system
#define MENUITEMS 5
#define F_CPU 16000000UL

// private method prototypes.
static uint32_t read_mem(uint32_t);
static void write_mem(uint32_t, uint32_t);
static void dump_mem(uint32_t, int);
static int check_addr(int);
static void printmenu();

const char* menuitem[] = {"read <addr>","write <addr> <val>","dump <addr>
<len>","help","quit"};

const char* helptext = "Type in menu selection followed by necessary
information."
            "\nFor example:"
            "\n2 0x20000000 16"
            "\nwill dump 16 bytes starting at address 0x20000000\n\n";

/**
 * The program is a console application which lets user to interact
 * with the memory.
 */
int main()
{
      init_usart2(19200, F_CPU);

      int cont = 1; // the variable for determining if the program should be
exit.
      char input[20]; // inputs from the user.
      char *token; // the command typed by user.
      int address, len, val, command;

      /* repeat displaying menu until cont = 0 */
      while(cont)
      {
            printmenu();

            fgets(input, 20, stdin); // gets input from console.
            token = strpbrk(input,"01234");
```

```
            /* Handle different commands */
            switch (token?*token:-1)
            {
                  // Read
                  case '0':
                        sscanf(input, "%d %x", &command, &address); // fetch
data from input.
                        if (check_addr(address)) {
                              int value = read_mem((uint32_t)address);
                              printf("0x%x\n", value);
                        }
                        break;
                  // Write
                  case '1':
                        val = 0;
                        sscanf(input, "%d %x %x", &command, &address, &val);
// fetch data from input.
                        if (check_addr(address)) {
                              write_mem(address, val);
                              printf("write %x with %x\n",address,val);
                        }
                        break;
                  // Dump
                  case '2':
                        sscanf(input, "%d %x %d", &command, &address, &len);
// fetch data from input.
                        if (check_addr(address)) {
                              dump_mem(address, len);
                              printf("\n");
                        }
                        break;
                  // Help
                  case '3':
                        printf("%s\n",helptext);
                        break;
                  // Exit
                  case '4':
                        printf("Exiting\n");
                        cont = 0;
                        break;
                  // Invalid selection
                  default:
                        printf("invalid selection\n");
                        break;
            }
      }

      return 0;
}

/**
 * The method prints all menu items.
 */
static void printmenu() {
      for(int i = 0;i<MENUITEMS;i++){
            printf("%d: %s\n",i,menuitem[i]);
```

```
	}
}

/**
 * The method reads 4 bytes from the given address.
 * Args:
 * address: the starting address for reading data.
 */
static uint32_t read_mem(uint32_t address) {
	volatile uint32_t* address_pointer = (uint32_t*) address;
	uint32_t value = *address_pointer;
	return value;
}

/**
 * The method writes 4 bytes to the given address.
 * Args:
 * address: the starting address for writing data.
 * value: the value needs to be written to the address.
 */
static void write_mem(uint32_t address, uint32_t value) {
	volatile uint32_t* address_pointer = (uint32_t*) address;
	*address_pointer = value;
}

/**
 * The method prints out a specified number of bytes from
 * the given memory.
 * Args:
 * address: the starting address.
 * length: how many bytes need to be printed.
 */
static void dump_mem(uint32_t address, int length) {
	volatile uint8_t* address_pointer = (uint8_t*) address;
	for (int i = 0; i < length; i++) {
		// determine if the next line is reached.
		if (i % 16 == 0) {
			printf("\n0x%x:", (int) address);
			address += 0x10;
		}
		uint8_t byte = address_pointer[i]; // get the current byte.
		printf(" ");
		// handle byte '0'.
		if (byte < 0x10) {
			printf("0");
		}
		printf("%x", byte);
	}
}

/**
 * Check the address if it is valid.
 * Args:
 * address: the address needs to be checked.
 */
static int check_addr(int address) {
	int valid = 1;
```

```
	if (address < 0) {
		valid = 0;
		printf("The address cannot be negative, and the number might be
overflow!\n");
	} else if (address % 4 != 0) {
		valid = 0;
		printf("The address is better be multiple of 4!\n");
	}
	return valid;
}
```