THE UNIVERSITY OF NEWCASTLE AUSTRALIA

School of Electrical Engineering and Computing

Dr Peter Stepien (peter.stepien@newcastle.edu.au)

Assignment 2

Instructions

- Complete all questions.
- Questions are not of equal value.
- To be completed in groups of two.
- Source code due Friday 4 May 2018 11:59pm.
- Late submissions will have 10% deducted per day.
- Demonstrate following week.

Assessment

Code will be assessed using the following criteria:

Component	*Unsatisfactory*	*Improving*	*Satisfactory*	*Excellent*
Specification (60%)	Does not meet specification	Partially meets specification	Meets specification, correct output	Novel extension implemented
Robustness (30%)	No error checking	Partial error checking	Handles provided errors, giving a warning message	Handles additional errors
Code Comments (10%)	No comments	Few comments	Mostly commented	Complete, with clear description of code

Use the template *System Workbench* projects provided to write your source code. Information about the templates and how to submit your source code on Blackboard is given on Page 4. The programs to be demonstrated to the tutors the week following the due date. The tutors will ask you to explain the operation code and may ask additional questions.

Objectives

The main objective is write an application within the constraints of an embedded system targeting the STM32 hardware. The application uses the graphics LCD display for user output and the associated touch panel is used for user input. The application also makes use of a serial connection.

The application to be implemented using the LCD display and touch panel input is a calculator. A command line interface and a number of commands is to be implemented using the serial connection. The serial interface will also be used to display debugging information for the calculator. Both the calculator function and the command line interface should run simultaneously on the STM32 hardware.

This assignment aims provides practice in writing an embedded application and highlights to issues regarding multiple input and the structure of the software. A looping executive will be used as the basis for the application, checking for user input and taking appropriate action.

The assignment contains two tasks to be completed. The first covers the *command line interface* and the second covers the *calculator*.

Question 1 (40 Marks)

The tasks requires the development of a *command line interface* over a serial connection. This is like a Unix *shell* or Microsoft Windows *command prompt*. This involves implementing a *command line parser* and a number of *commands*.

a) A *command line parser* is to be implemented using the serial connection between the STM32 hardware and a PC running PuTTY. The requirements of the command line parser is to:

- Echo keys typed on the PuTTY terminal program.
- Count the number of words that were typed.
- Extract the individual words from the command line.

An example of the command line parser is:

```
> command arg1 arg2
Count   : 3
Word(1) : command
Word(2) : arg1
Word(3) : arg2
>
```

Most of the work for this question has already been completed in Assignment 1. The example output from the command line parser will be replaced by the following commands to be implemented.

b) Implement *four numerical functions* using the command line parser:

- Add and multiply two or more real numbers.
- Subtract and divide two real numbers.

An example showing the result of commands that are entered on the command line interface are:

```
> add 1.1 2.2 3.3
Result : 6.6
> sub 2.3 1.2
Result : 1.1
> mul 2 3.1
Result : 6.2
> div 5.5 1.1
Result : 5
>
```

The software must be able to handle input that is not valid, such as replacing a number with a word.

c) Implement a command to turn *debug messages on and off* for Question 2:

- Turning it on will print useful messages while debugging code for Question 2.
- Turning it of will hide debug messages.

An example of the command operation is:

```
> debug on
Debug messages will be displayed.
> debug off
Debug messages will not be displayed.
>
```

Not shown are the debug messages that would be displayed when it is in the on state. The actual debug messages should be implemented as part of Question 2.

d) Implement a command to display *command help information*. The help command has an optional argument:

- Without any arguments, prints out help on all commands.
- With an argument, prints out help only on that command.

An example is:

```
> help div
div <num1> <num2> : Prints the result of num1/num2.
> help help
help [command] : Prints help information for command.
>
```

Details of the messages to be printed for each command are given in Table 1, which is a summary of the commands to be implemented.

Table 1: List of commands to be implemented

Command	*Description*
`add <num 1> .. <num N>`	Sum one or more numbers.
`sub <num 1> <num 2>`	Subtract two numbers.
`mul <num 1> .. <num N>`	Multiply one or more numbers.
`div <num 1> <num 2>`	Divide two numbers.
`debug <on|off>`	Turn debug messages on or off.
`help [<command>]`	Display help messages.

Feel free to experiment with the commands. This can included adding new commands or changing the functionality of the commands in Table 1. An example of an extended function would be to provide the current debug setting by using the `debug` command without an argument. An example of an extra function, would be to print out the number displayed on the calculator.

Question 2 (60 Marks)

Implement a simple *floating point calculator* using the STM32 hardware. The *touch panel* is to be used as input and the *LCD screen* is to be used as output. The calculator should have the following properties:

- Implement operations: add, subtract, multiply, divide.
- Implement keys: numbers, clear, decimal point, plus/minus.
- Display result in a format to fit the screen.

Make use of *debug messages* using the command line interface. These messages can be turned on and off using the `debug` command.

Details of the functions available to use the touch panel and LCD screen are given on on Page 5.

Use your creativity when designing the calculator!

Additional Information

The following provides information of writing your source code with *System Workbench* using the project templates provided and how to submit your source code and output transcript on Blackboard. An overview of the *Software Development Tools* is also available on Blackboard.

System Workbench Template Projects

The template projects are available on Blackboard in the folder containing resources for Assignment 2. There are two templates, one which targets Windows using MinGW and the other that targets the STM32 hardware, which are used for specific questions:

- `Ass-02-WINNT-rXXX.zip` : Question 1
- `Ass-02-STM32-rXXX.zip` : Question 1 and 2

Unzip the contents into the *System Workbench* workspace and import the projects.

The template projects are self contained and fully functional, with an example application which allows drawing onto the screen. This also provides a hint as to how to implement the keys for the calculator. Remove the example application code and replace it with your code.

The structure of the code uses a loop, with some initialisation code before the loop. This part is contained in the file `Ass-01.c`. This can be changed if required.

The following source files have been included for you to write code for each question:

- `Ass-02.h`
- `Ass-02-Q01.c`
- `Ass-02-Q02.c`
- `Ass-02-Lib.c`

The aim of the `Ass-02-Lib.c` file is to place functions that are common to other source files. Note that code can be used from Assignment 1. The include file requires updating when additional functions or data structures have been defined which are used in a number of source files.

The baud rate for the UART has been set to 115,200 bps. This will need to be set on PuTTY to correctly communicate with the STM32 hardware.

Submitting to Blackboard

The above listed source files form the deliverable for the assignment. A single file needs to be generated using the following from the command prompt in the folder with the source files:

- `type Ass-02*.c Ass-02.h > Ass-02-SID.txt`

Replace `SID` with your student number. You could also use `cat` instead of `type` under Linux. Note that the submitted file must be clear text.

Board Support Package Functions

Drivers are provided for peripherals that connect to the processor. These provide a Hardware Abstraction Layer (HAL) to allow software to be easily written across multiple platforms. These functions have difference names, including Board Support Package (BSP) functions.

The functions that can be used for the LCD display include:

```
uint8_t   BSP_LCD_Init(void);
uint32_t  BSP_LCD_GetXSize(void);
uint32_t  BSP_LCD_GetYSize(void);

uint16_t  BSP_LCD_GetTextColor(void);
uint16_t  BSP_LCD_GetBackColor(void);
void      BSP_LCD_SetTextColor(uint16_t Color);
void      BSP_LCD_SetBackColor(uint16_t Color);
void      BSP_LCD_SetFont(sFONT *fonts);
sFONT    *BSP_LCD_GetFont(void);

void      BSP_LCD_Clear(uint16_t Color);
void      BSP_LCD_ClearStringLine(uint16_t Line);
void      BSP_LCD_DisplayStringAtLine(uint16_t Line, uint8_t *ptr);
void      BSP_LCD_DisplayStringAt(uint16_t Xpos, uint16_t Ypos, uint8_t *Text,
                                  Line_ModeTypdef Mode);
void      BSP_LCD_DisplayChar(uint16_t Xpos, uint16_t Ypos, uint8_t Ascii);

uint16_t  BSP_LCD_ReadPixel(uint16_t Xpos, uint16_t Ypos);
void      BSP_LCD_DrawHLine(uint16_t Xpos, uint16_t Ypos, uint16_t Length);
void      BSP_LCD_DrawVLine(uint16_t Xpos, uint16_t Ypos, uint16_t Length);
void      BSP_LCD_DrawLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2);
void      BSP_LCD_DrawRect(uint16_t Xpos, uint16_t Ypos, uint16_t Width,
                           uint16_t Height);
void      BSP_LCD_DrawCircle(uint16_t Xpos, uint16_t Ypos, uint16_t Radius);
void      BSP_LCD_DrawPolygon(pPoint Points, uint16_t PointCount);
void      BSP_LCD_DrawEllipse(int Xpos, int Ypos, int XRadius, int YRadius);
void      BSP_LCD_DrawBitmap(uint16_t Xpos, uint16_t Ypos, uint8_t *pbmp);
void      BSP_LCD_DrawRGBImage(uint16_t Xpos, uint16_t Ypos, uint16_t Xsize,
                               uint16_t Ysize, uint8_t *pbmp);
void      BSP_LCD_FillRect(uint16_t Xpos, uint16_t Ypos, uint16_t Width,
                           uint16_t Height);
void      BSP_LCD_FillCircle(uint16_t Xpos, uint16_t Ypos, uint16_t Radius);
void      BSP_LCD_FillEllipse(int Xpos, int Ypos, int XRadius, int YRadius);

void      BSP_LCD_DisplayOff(void);
void      BSP_LCD_DisplayOn(void);
```

The functions that can be used for for the touch panel include:

```
uint8_t   BSP_TP_Init(void);
uint8_t   BSP_TP_GetDisplayPoint(Coordinate *pDisplay);
```

To read data from the UART directly, the following HAL function can be used:

```
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t
                                   *pData, uint16_t Size, uint32_t Timeout);
```

The `Timeout` argument is set to `0` to indicate that the function should not wait for a character to become available. See the template code for details of its use.

Some of the LEDs on the Discover Board can be used to provide visual feedback. They can be turned on and off using the following HAL functions:

```
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin,
                       GPIO_PinState PinState);
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
```

Some of the LEDs connect to pins that are used for the LCD display so can't be used. Two of the LEDs that can be used are connected to `LD3_Pin` and `LD4_Pin`.

Hints

To implement a command line interface, a function is normally associated with each command. The program parses the string entered via the terminal, extracts the words and checks whether the first word is a valid command. If it is a valid command, the appropriate function is called, using the additional arguments. Since the number of arguments in a function can vary, it is appropriate that the functions associated with the commands have a common prototype, such as the following:

```
int8_t XXXFunction(uint8_t ArgNum, uint8_t *ArgStrings[]);
```

Here, the first argument `ArgNum` is the number of arguments and `ArgStrings` is the array of argument strings. This is similar to the prototype of the `main()` function. Information for a command consists of the command string and the associated function. This can be stored using the following structure:

```
typedef struct{
  int8_t  *NameString;                          // Command string
  int8_t (*Function_p)(uint8_t ArgNum,          // Function pointer
                       uint8_t *ArgStrings[]); //
  int8_t  *HelpString;                          // Help information
} command_s;
```

An array of valid commands can then be defined as follows (just showing the numerical commands):

```
const command_s CommandList[] = {
  {"add", &AddFunction, "add␣<num␣1>␣..␣<num␣N>"},
  {"sub", &SubFunction, "sub␣<num␣1>␣<num␣2>"},
  {"mul", &MulFunction, "mul␣<num␣1>␣..␣<num␣N>"},
  {"div", &DivFunction, "div␣<num␣1>␣<num␣2>"},
  {NULL, NULL, NULL}
};
```

With the above definition `CommandList[]` is an array of objects of type `command_s`. Every element of `CommandList[]` has three members. For example, `CommandList[0].NameString` is a pointer to the string `"add"` and `CommandList[0].Function_p` is a function pointer to the function `AddFunction()` with a prototype:

```
  int8_t AddFunction(uint8_t ArgNum, uint8_t *ArgStrings[]);
```

Similarly the functions `SubFunction()`, `MulFunction()` and `DivFunction()` share the common prototype:

```
  int8_t SubFunction(uint8_t ArgNum, uint8_t *ArgStrings[]);
  int8_t MulFunction(uint8_t ArgNum, uint8_t *ArgStrings[]);
  int8_t DivFunction(uint8_t ArgNum, uint8_t *ArgStrings[]);
```

These functions are pointed to by `CommandList[1].Function_p`, `CommandList[2].Function_p` and `CommandList[3].Function_p`, respectively.

To use `CommandList[]`, write a function that steps through the array `CommandList[k]` and finds whether there is a `k` such that a given command string matches `CommandList[k].NameString`. If such a `k` exists, then the algorithm calls the function pointed to by `CommandList[k].Function_p`. Otherwise it responds that the function was not found.

References

There are a number of additional references that help understand the operation of the STM32 hardware and the library functions provided. In particular the HAL functions are used to access the STM32 microcontroller hardware, such as using the UART and toggling the LEDs.

While the drivers for the touch panel and LCD display are provided, you may want to understand how they they have been written and how it relates to the hardware. To do this, you need to look through the data sheets of the controllers used.

The following additional references are available on Blackboard:

- ST Microelectronics Documentation:
  - STM32F405xx STM32F407xx *Datasheet*
  - UM1472 User manual *Discovery kit with STM32F407VG MCU*
  - UM2052 User manual *Getting started with STM32 MCU Discovery Kits software development tools*
  - UM1725 User Manual *Description of STM32F4xx HAL drivers*
- Schematics:
  - Open407V-D *board schematic*
  - HY32D *LCD module schematic*
- Datasheets:
  - ILI9325 *a-Si TFT LCD single chip driver datasheet*
  - TSC2046 *Low voltage I/O touch screen controller datasheet*

The schematics are of interest as the indicate have the various devices connect back to the STM32 microcontroller. This also highlights the fact that there are some pins connect to more than one peripheral.