Functions (2J)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Based on Embedded Software in C for an ARM Cortex M
http://users.ece.utexas.edu/~valvano/Volume1/

Structure Declarations (1)

```
short FtoC(short TempF) {
    short TempC;

    TempC=(5*(TempF-32))/9;   // conversion
    return TempC;
}
```

Function Declarations

```
//  declaration                          input                 output
void    Ritual(void);                    // none               none
char    InChar(void);                    // none               8-bit
void    OutChar(char);                   // 8-bit              none
short   InSDec(void);                    // none               16-bit
void    OutSDec(short);                  // 16-bit             none
char    Max(char,char);                  // two 8-bit          8-bit
int     EMax(int,int);                   // two 32-bit         32-bit
void    OutString(char*);                // pointer to 8-bit   none
char *  alloc(int);                      // 32-bit             pointer to 8-bit
int     Exec(void(*fnctPt)(void));       // function pointer   32-bit
```

```
void    InitSCI(void);                   // Initialize 38400 bits/sec
char    InChar(void);                    // Reads in a character, gadfly
void    OutChar(char);                   // Output a character, gadfly
char    UpCase(char);                    // Converts lower case character to upper case
void    InString(char *, unsigned int);  // Reads in a String of max length
```

Function Declaration

```
void InitSCI();
char InChar();
void OutChar(char letter);
char UpCase(char letter);
void InString(char *pt, unsigned int MaxSize);
```

```
extern void InitSCI(void);
extern char InChar(void);
extern void OutChar(char);
extern char UpCase(char);
extern void InString(char *, unsigned int);
```

Function Pointers

```
int (*fp)(int);  // pointer to a function with input and output

int fun1(int input) {
   return(input+1);              // this adds 1
};

int fun2(int input){
   return(input+2);              // this adds 2
};

void Setp(void){ int data;
   fp = &fun1;                   // fp points to fun1
   data = (*fp)(5);              // data=fun1(5);

   fp = &fun2;                   // fp points to fun2
   data = (*fp)(5);              // data=fun2(5);
};
```

Function Definitions

```
#include "LCD.h"
#include "UART.H"
#include "SysTick.H"

void main(void) {
   char letter;
   short n=0;

   UART_Init();
   LCD_Init();
   SysTick_Init()
   LCD_String("This is a LCD");
   SysTick_Wait10ms(1000);
   LCD_clear();
   letter='a'-1;

   while(1){
      if (letter=='z')
         letter='a';
      else
         letter++;
      LCD_putchar(letter);
      SysTick_Wait10ms(250);

      if(++n==16){
         n=0;
         LCD_clear();
      }

   }
}
```

Functions Calls

```
#include "LCD.h"
#include "UART.H"
#include "SysTick.H"

void main(void) {
  char letter;
  short n=0;

  UART_Init();
  LCD_Init();
  SysTick_Init()
  LCD_String("This is a LCD");
  SysTick_Wait10ms(1000);
  LCD_clear();
  letter='a'-1;

  while(1){
    if (letter=='z')
      letter='a';
    else
      letter++;

    LCD_putchar(letter);
    SysTick_Wait10ms(250);

    if(++n==16){
      n=0;
      LCD_clear();
    }
  }
}
```

Argument Passing

```
int GetFifo (char *datapt) {
  if(Size == 0 )
    return(0);                                  /* Empty if Size=0 */
  else{
    *datapt=Fifo[GetI++]; Size--;
    if (GetI == FifoSize) GetI = 0;
    return(-1);
  }
}

char InChar(void) {
  char data;
  while(GetFifo(&data)) { };
  return (data);
}
```

Argument Passing

```
int PutFifo(char data) {
  if(Size == FifoSize ) {
     return(0);                                          /* Failed, fifo was full */
  }
  else{
    Size++;
    *(PutPt++)=data;                                     /* put data into fifo */
    if (PutPt == &Fifo[FifoSize]) PutPt = &Fifo[0];      /* Wrap */
    return(-1);                                          /* Successful */
  }
}

void OutChar(char data){
   while(PutFifo(data)) { };
}

void main(void){ char data=0x41;
   OutChar(data);
}
```

Private vs Public Functions

```
unsigned long static TimerClock;                          // private global

void SysTick_Init(void) {
  NVIC_ST_CTRL_R = 0;                                     // 1) disable SysTick during setup
  NVIC_ST_RELOAD_R = 0x00FFFFFF;                          // 2) maximum reload value
  NVIC_ST_CURRENT_R = 0;                                  // 3) any write to current clears it
  NVIC_ST_CTRL_R = 0x00000005;                            // 4) enable SysTick with core clock
}

// The delay parameter is in units of the 80 MHz core clock. (12.5 ns)
void static SysTick_Wait(unsigned long delay) {
  NVIC_ST_RELOAD_R = delay-1;                             // number of counts to wait
  NVIC_ST_CURRENT_R = 0;                                  // any value written to CURRENT clears
  while((NVIC_ST_CTRL_R & 0x00010000)==0) { }             // wait for count flag
}

                                                          // 10000us equals 10ms
void SysTick_Wait10ms(unsigned long delay){
  unsigned long i;

  for(i=0; i<delay; i++){
    SysTick_Wait(800000);                                 // wait 10ms
  }
}
```

Finite State Machine

```
struct State {
   void (*CmdPt)(void);                  /* function to execute */
   unsigned short Wait;                  /* Time, 10ms  to wait */
   unsigned char AndMask[4];
   unsigned char EquMask[4];
   const struct State *Next[4];          /* Next states */
};

typedef const struct State state_t;
typedef state_t; * StatePtr;

#define stop &fsm[0]
#define turn &fsm[1]
#define bend &fsm[2]

void DoStop(void){  PORTA = 0x34;}
void DoTurn(void){  PORTA = 0xB3;}
void DoBend(void){  PORTA = 0x75;}
```

```
state_t fsm[3]={
{&DoStop, 2000,                          // stop 1 ms
  {0xFF,  0xF0,  0x27,  0x00},
  {0x51,  0xA0,  0x07,  0x00},
  {turn,  stop,  turn,  bend}},
{&DoTurn,5000,                           // turn 2.5 ms
  {0x80,  0xF0,  0x00,  0x00},
  {0x00,  0x90,  0x00,  0x00},
  {bend,  stop,  turn,  turn}},
{&DoBend,4000,                           // bend 2 ms
  {0xFF,  0x0F,  0x01,  0x00},
  {0x12,  0x05,  0x00,  0x00},
  {stop,  stop,  turn,  stop}}};
```

Linked Lists

```
void control(void) {
  StatePtr Pt;
  unsigned char Input;
  unsigned short startTime;
  unsigned int i;

  SysTick_Init();
  Port_Init();

  Pt = stop;                                          // Initial State
  while(1){
    (*Pt->CmdPt)();                                   // 1) execute function
    SysTick_Wait10ms(Pt->Wait);                       // 2) wait
    Input = PORTB;                                    // 3) input
    for(i=0;i<4;i++)
      if((Input&Pt->AndMask[i])==Pt->EquMask[i]){
        Pt=Pt->Next[i];                               // 4) next depends on input
        i=4;
      }
  }
}
```

Linked Lists

```
// Linked List Interpreter
struct Node{
    unsigned char Letter;
    void (*fnctPt)(void);
    const struct Node *Next;
};

typedef const struct Node node_t;
typedef node_t * NodePtr;

void CommandA(void){
    OutString("\nExecuting Command a");
}

void CommandB(void){
    OutString("\nExecuting Command b");
}

void CommandC(void){
    OutString("\nExecuting Command c");
}

node_t LL[3]={
    { 'a', &CommandA, &LL[1]},
    { 'b', &CommandB, &LL[2]},
    { 'c', &CommandC, 0 }};
```

Inserting (1)

```
void main(void){
   NodePtr Pt;
   char string[40];

   UART_Init();                                    // Enable SCI port
   UART_OutString("\nEnter a single letter command followed by <enter>");

   while(1){
      UART_OutString("\n>");
      UART_InString(string,39);                    // first character is interpreted
      Pt=&LL[0];                                   // first node to check
      while(Pt){
         if(string[0]==Pt->Letter){
            Pt->fnctPt();                          // execute function
            break;}                                // leave while loop
         else{
            Pt=Pt->Next;
            if(Pt==0) UART_OutString(" Error");
         }
      }
   }

}
```

References

[1] Essential C, Nick Parlante
[2] Efficient C Programming, Mark A. Weiss
[3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
[4] C Language Express, I. K. Chun
[5] “A Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux“
http://cseweb.ucsd.edu/~ricko/CSE131/teensyELF.htm
[6] http://en.wikipedia.org
[7] http://www.muppetlabs.com/~breadbox/software/tiny/teensy.html
[8] http://csapp.cs.cmu.edu/public/ch7-preview.pdf