Structures (2I)

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

This document was produced by using LibreOffice.

Young Won Lim
4/17/18

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

Structure Declarations (1)

```
struct theport {
   unsigned char               mask;       // defines which bits are active
   unsigned long volatile *    addr;       // pointer to its address
   unsigned long volatile *    ddr;        // pointer to its direction reg
};

struct theport PortA, PortB, PortE;

struct theport {
   unsigned char               mask;       // defines which bits are active
   unsigned long volatile *    addr;
   unsigned long volatile *    ddr;
} PortA, PortB, PortE;
```

Structure Declarations (2)

```
struct theport {
   unsigned char                mask;       // defines which bits are active
   unsigned long volatile *     addr;       // address
   unsigned long volatile *     ddr;        // direction reg
};
typedef struct theport port_t;
port_t PortA, PortB, PortE;
```

```
typedef struct {
   unsigned char                mask;       // defines which bits are active
   unsigned long volatile *     addr;       // address
   unsigned long volatile *     ddr;        // direction reg
} port_t;
port_t PortA, PortB, PortE;
```

```
struct port {
   unsigned char mask;                      // defines which bits are active
   unsigned long volatile *     addr;       // address
   unsigned long volatile *     ddr;        // direction reg
};
typedef struct port port;
port PortA, PortB, PortE;
```

Accessing Members

```
PortB.mask      = 0xFF;                                  // the TM4C123 has 8 bits on PORTB
PortB.addr      = (unsigned long volatile *)  (0x400053FC);
PortB.ddr       = (unsigned long volatile *)  (0x40005400);

PortE.mask      = 0x3F;                                  // the TM4C123 has 6 bits on PORTE
PortE.addr      = (unsigned long volatile *)  (0x400243FC);
PortE.ddr       = (unsigned long volatile *)  (0x40024400);

(*PortE.ddr)    = 0;                                     // specify PortE as inputs
(*PortB.addr)   = (*PortE.addr);                         // copy from PortE to PortB
```

Accessing Members

```
struct theline {
   int  x1,y1;                  // starting point
   int  x2,y2;                  // starting point
   unsigned char color;         // color
};

typedef struct theline line_t;

struct thepath {
   line_t L1,L2;                // two lines
   char direction;
};

typedef struct thepath path_t;
path_t p;                       // global
```

```
void Setp(void) {
   line_t     myLine;
   path_t     q;

   p.L1.x1 = 5;     // black line from 5,6 to 10,12
   p.L1.y1 = 6;
   p.L1.x2 = 10;
   p.L1.y2 = 12;
   p.L1.color = 255;

   p.L2.x1 = 0;     // white line from 0,1 to 2,3
   p.L2.y1 = 1;
   p.L2.x2 = 2;
   p.L2.y2 = 3;
   p.L2.color = 0;

   p.direction = -1;
   myLine = p.L1;
   q = p;
};
```

Initializations

```
path_t   thePath ={    {0,0,5,6,128},     {5,6,-10,6,128},   1      };
line_t   theLine ={    0,0,5,6,128        };
port_t   PortE   ={    0x3F,
                       (unsigned long volatile *) (0x400243FC),
                       (unsigned long volatile *) (0x40024400)        };

path_t   thePath ={    {0,0,5,6,128},     };
line_t   thePath ={    5,6,10,12,         };
port_t   PortE   ={    1,  (unsigned char volatile *) (0x100A),       };
```

Initialization (2)

```
struct State {
   unsigned char        Out;          /* Output to Port B */
   unsigned short       Wait;         /* Time (62.5ns cycles) 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]

state_t fsm[3]={ {0x34,  16000,          // stop 1 ms
                 {0xFF,  0xF0,  0x27,  0x00},
                 {0x51,  0xA0,  0x07,  0x00},
                 {turn,   stop,  turn,  bend} },
               { {0xB3,  40000,          // turn 2.5 ms
                 {0x80,  0xF0,  0x00,  0x00},
                 {0x00,  0x90,  0x00,  0x00},
                 {bend,  stop,  turn,  turn} },
               { {0x75,  32000,          // bend 2 ms
                 {0xFF,  0x0F,  0x01,  0x00},
                 {0x12,  0x05,  0x00,  0x00},
                 {stop,   stop,  turn,  stop} }   };
```

Using Pointers

```
void Setp(void) {
  path_t *ppt;

  ppt = &p;                  // pointer to an existing global variable
  ppt->L1.x1 = 5;            // black line from 5,6 to 10,12
  ppt->L1.y1 = 6;
  ppt->L1.x2 = 10;
  ppt->L1.y2 = 12;
  ppt->L1.color = 255;

  ppt->L2.x1 = 0;            // white line from 0,1 to 2,3
  ppt->L2.y1 = 1;
  ppt->L2.x2 = 2;
  ppt->L2.y2 = 3;
  ppt->L2.color = 0;
  ppt->direction = -1;

  (*ppt).direction = -1;
};
```

Finite State Machine

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

  SysTick_Init();
  Port_Init();
  Pt = stop;                                          // Initial State
  while (1) {
    GPIO_PORTA_DATA_R = Pt->Out;                      // 1) output
    SysTick_Wait(Pt->Wait);                           // 2) wait
    Input = GPIO_PORTB_DATA_R;                        // 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;
      }
    }
};
```

Passing Structures

```
typedef const struct {
  unsigned char mask;              // defines which bits are active
  unsigned long volatile *addr;    // address
  unsigned long volatile *ddr;     // direction reg
} port;

port_t PortE= {0x3F,
   (unsigned long volatile *) (0x400243FC),
   (unsigned long volatile *) (0x40024400)
};

port_t PortF={0x1F,
  (unsigned long volatile *) (0x400253FC),
  (unsigned long volatile *) (0x40025400)
};

int MakeOutput(port_t *ppt) {
  (*ppt->ddr) = ppt->mask;         // make output
  return 1;
}

int MakeInput(port_t *ppt) {
  (*ppt->ddr )= 0x00;              // make input
  return 1;
}
```

```
unsigned char Input( port_t *ppt) {
   return (*ppt→addr);
}

void Output(port_t *ppt, unsigned char data) {
   (*ppt->addr) = data;
}

int main(void) {
  unsigned char MyData;

  MakeInput(&PortE);
  MakeOutput(&PortF);
  Output(&PortF,0);
  MyData=Input(&PortE);
  return 1;
}
```

Linked Lists

```
struct node {
  unsigned short data;          // 16 bit information
  struct node *next;            // pointer to the next
};
typedef struct node node_t;
node_t *HeadPt;
```

```
#include <stdlib.h>;
int StoreData(unsigned short info) {
   node_t *pt;

   pt=malloc(sizeof(node_t));   // create a new entry
   if (pt) {
     pt->data=info;             // store data
     pt->next=HeadPt;           // link into existing
     HeadPt=pt;
     return(1);
   }
   return(0);                   // out of memory
}
```

Linked Lists (2)

```
node_t *Search(unsigned short info) {
  node_t *pt;

  pt=HeadPt;
  while (pt) {
    if (pt->data==info)
        return (pt);
    pt=pt->next;                    // link to next
  }
  return(pt);                       // not found
};


unsigned short Count(void) {
  node_t *pt;
  unsigned short cnt;

  cnt = 0;
  pt = HeadPt;
  while (pt) {
    cnt++;
    pt = pt->next;                  // link to next
  }
  return(cnt);
};
```

Inserting (1)

```
int InsertData(unsigned short info) {
   node_t *firstPt,*secondPt,*newPt;
   newPt = malloc(sizeof(node_t));          // create a new entry

   if (newPt) {
     newPt->data = info;                    // store data
     if (HeadPt==0) {                       // case 1
       newPt->next = HeadPt;                // only element
       HeadPt = newPt;
       return(1);
     }

     if (info<=HeadPt->data) {              // case 2
       newPt->next = HeadPt;                // first element in list
       HeadPt = newPt;
       return(1);
     }
```

Inserting (1)

```
    firstPt = HeadPt;                   // search from beginning
    secondPt = HeadPt->next;

    while (secondPt) {
      if (info <= secondPt->data) {     // case 3
        newPt->next = secondPt;         // insert element here
        firstPt->next = newPt;
        return(1);
      }

      firstPt = secondPt;               // search next
      secondPt = secondPt->next;
    }

    newPt->next = secondPt;             // case 4, insert at end
    firstPt->next = newPt;
    return(1);
  }
  return(0);                            // out of memory
};
```

Deleting

```
int Remove(unsigned short info) {
  node_t *firstPt,*secondPt;

  if (HeadPt==0)                              // case 1
    return(0);                                // empty list
  firstPt = HeadPt;
  secondPt = HeadPt->next;

  if (info==HeadPt→data) {                    // case 2
    HeadPt = secondPt;                        // remove first element in list
    free(firstPt);                            // return unneeded memory to heap
    return(1);
  }

  while (secondPt) {
    if (secondPt->data==info) {               // case 3
      firstPt->next=secondPt->next;           // remove this one
      free(secondPt);                         // return unneeded memory to heap
      return(1);
    }
    firstPt = secondPt;                       // search next
    secondPt = secondPt->next;
  }

  return(0);                                   // case 4, not found
};
```

Huffman Code (1)

```
struct Node  {
    char Letter0;                   // ASCII code if binary 0
    char Letter1;                   // ASCII code if binary 1
                                    // Letter1 is NULL(0) if Link is pointer to another node
    const struct Node *Link;        // binary tree pointer
};

typedef const struct Node node_t;
```

Huffman Code (1)

```
// Huffman tree
node_t twentysixth      = {'Q',  'Z',  0};
node_t twentyfifth      = {'X',  0,    &twentysixth};
node_t twentyfourth     = {'G',  0,    &twentyfifth};
node_t twentythird      = {'J',  0,    &twentyfourth};
node_t twentysecond     = {'W',  0,    &twentythird};
node_t twentyfirst      = {'V',  0,    &twentysecond};
node_t twentyth         = {'H',  0,    &twentyfirst};
node_t ninteenth        = {'F',  0,    &twentyth};
node_t eighteenth       = {'B',  0,    &ninteenth};
node_t seventeenth      = {'K',  0,    &eighteenth};
node_t sixteenth        = {'D',  0,    &seventeenth};
node_t fifteenth        = {'P',  0,    &sixteenth};
node_t fourteenth       = {'M',  0,    &fifteenth};
node_t thirteenth       = {'Y',  0,    &fourteenth};
node_t twelfth          = {'L',  0,    &thirteenth};
node_t eleventh         = {'U',  0,    &twelfth};
node_t tenth            = {'R',  0,    &eleventh};
node_t ninth            = {'C',  0,    &tenth};
node_t eighth           = {'O',  0,    &ninth};
node_t seventh          = {' ',  0,    &eighth};
node_t sixth            = {'N',  0,    &seventh};
node_t fifth            = {'I',  0,    &sixth};
node_t fourth           = {'S',  0,    &fifth};
node_t third            = {'T',  0,    &fourth};
node_t second           = {'A',  0,    &third};
node_t root             = {'E',  0,    &second};
```

Huffman Code (1)

```
//********encode***************
// convert ASCII string to Huffman bit sequence
// returns bit count       if OK
// returns 0               if BitFifo Full
// returns 0xFFFF          if illegal character


//********decode***************
// convert Huffman bit sequence to ASCII
// will remove from the BitFifo until it is empty
// returns character count
```

Huffman Code (1)

```
int encode(char *sPt) {                  // null-terminated ASCII string
 int NotFound;  char data;
 int BitCount = 0;                       // number of bits created
 node_t *hpt;                            // pointer into Huffman tree

  while (data = (*sPt)) {
    sPt++;                               // next character
    hpt = &root;                         // start search at root
    NotFound = 1;                        // changes to 0 when found
    while (NotFound) {
      if ((hpt->Letter0) =  = data)  {
        if (!BitPut(0))    return (0);   // data structure full
        BitCount++;
        NotFound = 0;
      } else {
        if (!BitPut(1))   return (0);    // data structure full
        BitCount++;
        if ((hpt->Letter1) =  = data)
          NotFound = 0;
        else {                           // doesn't match either Letter0 or Letter1
          hpt = hpt->Link;
          if (hpt == 0) return (0xFFFF); // illegal, end of tree?
        }
      }
    }
 }
 return BitCount;
}
```

Huffman Code (1)

```
int decode(char *sPt) {                 // null-terminated ASCII string
 int CharCount=0;                       // number of ASCII characters created
 unsigned int data;
 node_t *hpt;                           // pointer into Huffman tree
 hpt=&root;                             // start search at root

 while (BitGet(&data)) {
  if (data==0) {
    (*sPt)= (hpt->Letter0);
    sPt++;
    CharCount++;
    hpt=&root;
  }                                     // start over and search at root
  else                                  //data is 1
    if ((hpt→Link)==0) {
      (*sPt)= (hpt->Letter1);
      sPt++;
      CharCount++;
      hpt=&root;
     }                                  // start over and search at root
    else {                              // doesn't match either Letter0 or Letter1
      hpt=hpt->Link;
    }
  }
  (*sPt)=0;                             // null terminated
  return CharCount;
}
```

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