Concise Embedded C

Embedded System 2561, KU CSC

Adapted by Sorayut Glomglome

C programming for embedded microcontroller systems.

Assumes experience with assembly language programming.

V. P. Nelson

Outline

- Program organization and microcontroller memory
- Data types, constants, variables
- Microcontroller register/port addresses
- Operators: arithmetic, logical, shift
- Control structures: if, while, for
- Functions
- Interrupt routines

Basic C program structure

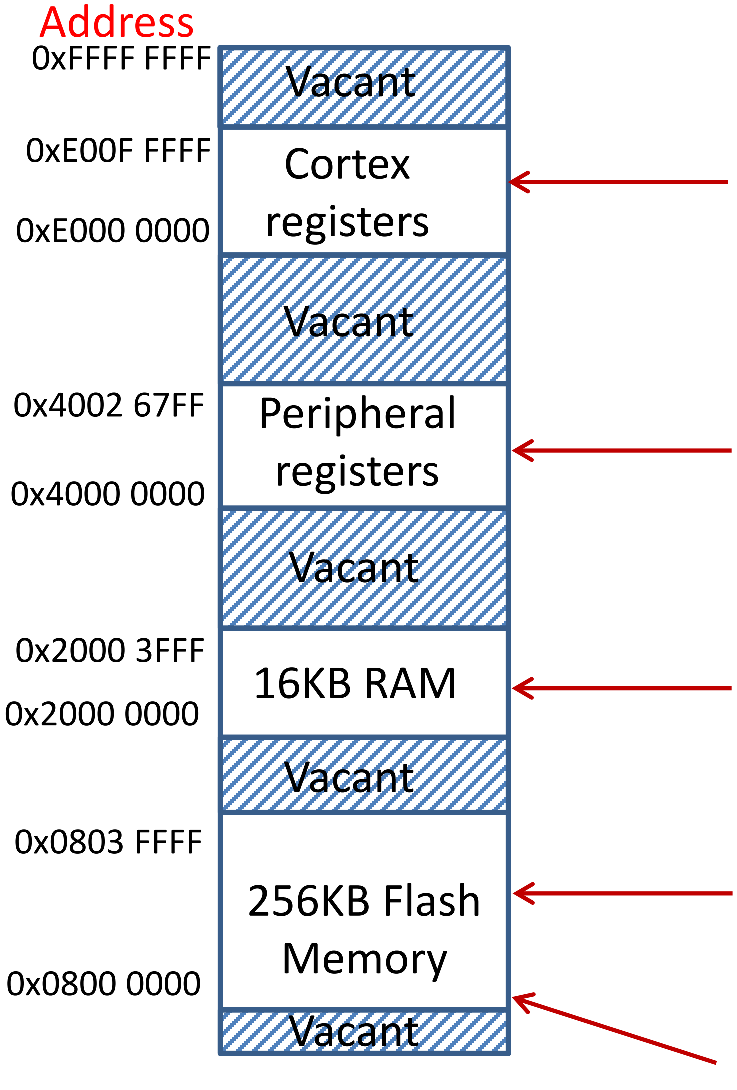
```
#include "STM32L1xx.h"      /* I/O port/register names/addresses for the STM32L1xx microcontrollers */
/* Global variables – accessible by all functions */
int count, bob;             //global (static) variables – placed in RAM
/* Function definitions*/
int function1(char x) {     //parameter x passed to the function, function returns an integer value
  int i,j;                  //local (automatic) variables – allocated to stack or registers
  -- instructions to implement the function
}
/* Main program */
void main(void) {
  unsigned char sw1;        //local (automatic) variable (stack or registers)
  int k;                    //local (automatic) variable (stack or registers)
/* Initialization section */
  -- instructions to initialize  variables, I/O ports, devices, function registers
/* Endless loop */
  while (1) {          //Can also use:  for(;;) {
   -- instructions to be repeated
  } /* repeat forever */
}
```

Declare local variables

Initialize variables/devices

Body of the program

STM32L100RC µC memory map



Control/data registers: Cortex-M3 CPU functions
(NVIC, SysTick Timer, etc.)

PORT A จะอยู่ในพื้นที่นี้ด้วย

Control/data registers: microcontroller peripherals
(timers, ADCs, UARTs, etc.) (อุปกรณ์ต่อพ่วง)

16K byte RAM: variable & stack storage

256K byte Flash memory:
program code & constant data storage

Reset & interrupt vectors: in 1st words of flash memory

Microcontroller “header file”

- *Keil MDK-ARM* provides a *derivative-specific* “header file” for each microcontroller, which defines memory addresses and symbolic labels for CPU and peripheral function register addresses.

```
#include "STM32L1xx.h“          /* target uC information */

// GPIOA configuration/data register addresses are defined in STM32L1xx.h
void main(void) {
  uint16_t  PAval;                             //16-bit unsigned variable
  GPIOA->MODER    &=  ~(0x00000003);           // Set GPIOA pin PA0 as input
  PAval = GPIOA->IDR;                          // Set PAval to 16-bits from GPIOA
  for(;;) {}          /* execute forever */    // 0xFFFFFFFC => ...1111 1100
}                                              // 00   คือ input mode ดูจากสไลด์ GPIO
```

C compiler data types

- Always match data type to data characteristics!
- Variable type indicates how data is represented
  - #bits determines range of numeric values
  - signed/unsigned determines which arithmetic/relational operators are to be used by the compiler
  - non-numeric data should be "unsigned"
- Header file "stdint.h" defines alternate type names for standard C data types
  - Eliminates ambiguity regarding #bits
  - Eliminates ambiguity regarding signed/unsigned

(Types defined on next page)

C compiler data types

Data type declaration *	Number of bits	Range of values

char k;
unsigned char k;
uint8_t k;
	8	0..255

signed char k;
int8_t k;
	8	-128..+127

short k;
signed short k;
int16_t k;
	16	-32768..+32767

unsigned short k;
uint16_t k;
	16	0..65535

int k;
signed int k;
int32_t k;
	32	-2147483648..+2147483647

unsigned int k;
uint32_t k;
	32	0..4294967295

* intx_t and uintx_t defined in *stdint.h*

Data type examples

- Read bits from GPIOA (16 bits, non-numeric)
  - *uint16_t n; n = GPIOA->IDR; //or: unsigned short n;*
- Write TIM2 prescale value (16-bit unsigned)
  - *uint16_t t; TIM2->PSC = t; //or: unsigned short t;*
- Read 32-bit value from ADC (unsigned)
  - *uint32_t a; a = ADC; //or: unsigned int a;*
- System control value range [-1000…+1000]
  - *int32_t ctrl; ctrl = (x + y)*z; //or: int ctrl;*
- Loop counter for 100 program loops (unsigned)
  - *uint8_t cnt; //or: unsigned char cnt;*
  - *for (cnt = 0; cnt < 20; cnt++) {*

Constant/literal values

- Decimal เลขฐาน 10 ระบุเลขตามปกติ

```
int m,n;              //16-bit signed numbers
m = 453;   n = -25;
```

- Hexadecimal: ขึ้นต้นด้วย 0x หรือ 0X เป็นเลขฐาน 16

```
m = 0xF312; n = -0x12E4;
```

- Octal: ขึ้นต้นด้วยเลขศูนย์ (0) เป็นเลขฐาน 8

```
m = 0453; n = -023;
```

ห้ามใช้เลข 0 นำหน้าในตัวแปลเลขฐาน 10 (Decimal) เพราะมันจะถูกมองว่าเป็นเลขฐาน 8.

- Character: 1 ตัวอักษรใน single quotes, หรือ ค่า ASCII ตัวอักษรที่ขึ้นต้นด้วย “backslash”

```
m = ‘a’;       //ASCII value 0x61
n = ‘\13’;     //ASCII value 13 is the “return” character
```

- String (array) of characters:

```
unsigned char k[7];    // ควรประกาศตัวแปลให้เก็บตัวอักษรให้มากขึ้น 1 ค่า เพื่อเก็บค่า '\0'
strcpy(k,“hello\n”);   //k[0]=‘h’, k[1]=‘e’, k[2]=‘l’, k[3]=‘l’, k[4]=‘o’,
                       //k[5]=13 or ‘\n’ (ASCII new line character),
                       //k[6]=0 or ‘\0’ (null character – end of string)
```

Program variables

- A *variable* is an addressable storage location to information to be used by the program
  - Each variable must be *declared* to indicate size and type of information to be stored, plus name to be used to reference the information

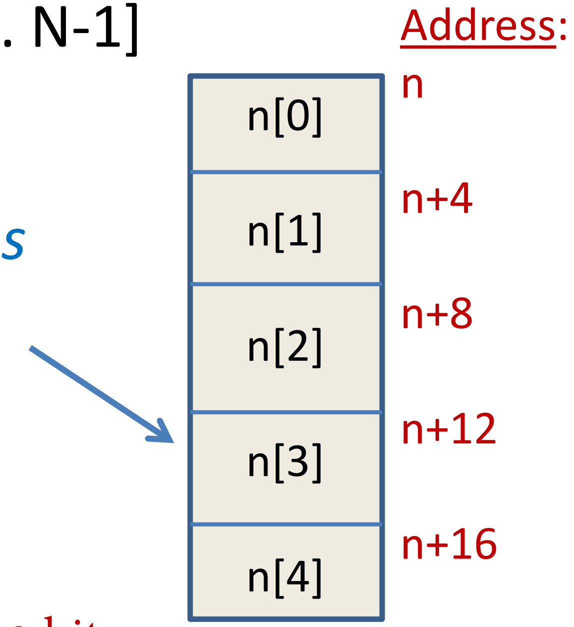
    int x,y,z;   //declares 3 variables of type "int"

    char a,b;   //declares 2 variables of type "char"
  - Space for variables may be allocated in registers, RAM, or ROM/Flash (for constants)
  - Variables can be *automatic* or *static*

Variable arrays

- An *array* is a set of data, stored in consecutive memory locations, beginning at a named address
  - Declare array name and number of data elements, N
  - Elements are "indexed", with indices [0 .. N-1]

```
int n[5];       //declare array of 5 "int" values
n[3] = 5;       //set value of 4th array element
```



Address:

n[0]	n
n[1]	n+4
n[2]	n+8
n[3]	n+12
n[4]	n+16

Note: Index of first element is always 0.

ค่า Address เพิ่มขึ้นทีละ 4 เพราะ int ใช้พื้นที่ 4 byte หรือ 32 bit

Static variables

static variables เป็นค่าที่ประกาศใช้ทั่วๆไป

- Retained for use throughout the program in RAM locations that are *not reallocated* during program execution.
- Declare either within or outside of a function
  - If declared outside a function, the variable is *global* in scope, i.e. known to all functions of the program
    - Use “normal” declarations. Example: *int count;*
  - If declared within a function, insert key word *static* before the variable definition. The variable is *local* in scope, i.e. known only within this function.

```
static unsigned char bob;
static int pressure[10];
```

static ที่ประกาศในฟังก์ชัน คือค่าที่ประกาศใช้เฉพาะในฟังก์ชันนั้น ฟังก์ชันอื่นใช้ไม่ได้
โดยปกติ ค่าที่ประกาศด้วย static ที่ไม่ได้กำหนดค่าเริ่มต้นมันจะทำการ initialize เป็น 0 ก่อนเสมอ

Static variable example

```
unsigned char count;  //global variable is static – allocated a fixed RAM location
                      //count can be referenced by any function
void math_op () {
 int i;               //automatic variable – allocated space on stack when function entered
 static int j;        //static variable – allocated a fixed RAM location to maintain the value
 if (count == 0)      //test value of global variable count
    j = 0;            //initialize static variable j first time math_op() entered
 i = count;           //initialize automatic variable i each time math_op() entered
 j = j + i;           //change static variable j – value kept for next function call
}                     //return & deallocate space used by automatic variable i

void main(void) {
 count = 0;           //initialize global variable count
 while (1) {
  math_op();
  count++;            //increment global variable count
 }
}
```

C statement types

- Simple variable assignments
  - Includes input/output data transfers
- Arithmetic operations
- Logical/shift operations
- Control structures
  - IF, WHEN, FOR, SELECT
- Function calls
  - User-defined and/or library functions

Arithmetic operations

- C examples – with standard arithmetic operators

```
int i, j, k;     // 32-bit signed integers
uint8_t m,n,p;   // 8-bit unsigned numbers
i = j + k;       // add 32-bit integers
m = n - 5;       // subtract 8-bit numbers
j = i * k;       // multiply 32-bit integers
m = n / p;       // quotient of 8-bit divide
m = n % p;       // remainder of 8-bit divide
i = (j + k) * (i – 2);  //arithmetic expression
```

*, /, % are higher in precedence than +, - (higher precedence applied 1st)

Example: j * k + m / n = (j * k) + (m / n)

Floating-point formats are not directly supported by Cortex-M3 CPUs.

Bit-parallel logical operators

Bit-parallel (bitwise) logical operators produce n-bit results of the corresponding logical operation:

& (AND) | (OR)   ^ (XOR)   ~ (Complement)

```
C = A & B;          A   0 1 1 0 0 1 1 0
(AND)               B   1 0 1 1 0 0 1 1
                    C   0 0 1 0 0 0 1 0

C = A | B;          A   0 1 1 0 0 1 0 0
(OR)                B   0 0 0 1 0 0 0 0
                    C   0 1 1 1 0 1 0 0

C = A ^ B;          A   0 1 1 0 0 1 0 0
(XOR)               B   1 0 1 1 0 0 1 1
                    C   1 1 0 1 0 1 1 1

B = ~A;             A   0 1 1 0 0 1 0 0
(COMPLEMENT)        B   1 0 0 1 1 0 1 1
```

Bit set/reset/complement/test

- Use a “mask” to select bit(s) to be altered

```
C = A & 0xFE;     A    a b c d e f g h
               0xFE    1 1 1 1 1 1 1 0   Clear selected bit of A
                  C    a b c d e f g 0

C = A & 0x01;     A    a b c d e f g h
               0xFE    0 0 0 0 0 0 0 1   Clear all but the selected bit of A
                  C    0 0 0 0 0 0 0 h

C = A | 0x01;     A    a b c d e f g h
               0x01    0 0 0 0 0 0 0 1   Set selected bit of A
                  C    a b c d e f g 1

C = A ^ 0x01;     A    a b c d e f g h
              0x01     0 0 0 0 0 0 0 1   Complement selected bit of A
                  C    a b c d e f g h’
```

Bit examples for input/output

- Create a “pulse” on bit 0 of PORTA (assume bit is initially 0)

```
PORTA = PORTA | 0x01;   //Force bit 0 to 1
PORTA = PORTA & 0xFE;  //Force bit 0 to 0
```

- Examples:

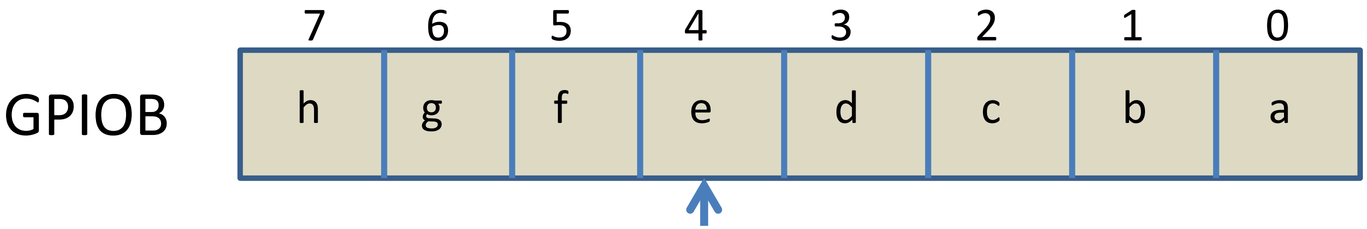
```
 if ( (PORTA & 0x80) != 0 )  //Or: ((PORTA & 0x80) == 0x80)
     bob();                  // call bob() if bit 7 of PORTA is 1
c = PORTB & 0x04;            // mask all but bit 2 of PORTB value
if ((PORTA & 0x01) == 0)     // test bit 0 of PORTA
    PORTA = c | 0x01;        // write c to PORTA with bit 0 set to 1
```

Example of µC register address definitions in *STM32Lxx.h*

(read this header file to view other peripheral functions)

```
#define PERIPH_BASE          ((uint32_t)0x40000000)       //Peripheral base address in memory
#define AHBPERIPH_BASE       (PERIPH_BASE + 0x20000)      //AHB peripherals
/* Base addresses of blocks of GPIO control/data registers */
#define GPIOA_BASE           (AHBPERIPH_BASE + 0x0000)    //Registers for GPIOA
#define GPIOB_BASE           (AHBPERIPH_BASE + 0x0400)    //Registers for GPIOB
#define GPIOA             ((GPIO_TypeDef *) GPIOA_BASE)   //Pointer to GPIOA register block
#define GPIOB             ((GPIO_TypeDef *) GPIOB_BASE)   //Pointer to GPIOB register block
/* Address offsets from GPIO base address – block of registers defined as a “structure” */
typedef struct
{
  __IO uint32_t MODER;     /*!< GPIO port mode register,                     Address offset: 0x00       */
  __IO uint16_t OTYPER;    /*!< GPIO port output type register,              Address offset: 0x04       */
  uint16_t RESERVED0;      /*!< Reserved,                                                    0x06       */
  __IO uint32_t OSPEEDR;   /*!< GPIO port output speed register,             Address offset: 0x08       */
  __IO uint32_t PUPDR;     /*!< GPIO port pull-up/pull-down register,        Address offset: 0x0C       */
  __IO uint16_t IDR;       /*!< GPIO port input data register,               Address offset: 0x10       */
  uint16_t RESERVED1;      /*!< Reserved,                                                    0x12       */
  __IO uint16_t ODR;       /*!< GPIO port output data register,              Address offset: 0x14       */
  uint16_t RESERVED2;      /*!< Reserved,                                                    0x16       */
  __IO uint16_t BSRRL;     /*!< GPIO port bit set/reset low registerBSRR,    Address offset: 0x18       */
  __IO uint16_t BSRRH;     /*!< GPIO port bit set/reset high registerBSRR,   Address offset: 0x1A       */
  __IO uint32_t LCKR;      /*!< GPIO port configuration lock register,       Address offset: 0x1C       */
  __IO uint32_t AFR[2];    /*!< GPIO alternate function low register,        Address offset: 0x20-0x24 */
} GPIO_TypeDef;
```

Example: I/O port bits (using bottom half of GPIOB)



	7	6	5	4	3	2	1	0
GPIOB	h	g	f	e	d	c	b	a

Switch connected to bit 4 (PB4) of GPIOB

```
uint16_t sw;                 //16-bit unsigned type since GPIOB IDR and ODR = 16 bits
sw = GPIOB->IDR;             // sw = xxxxxxxxhgfedcba  (upper 8 bits from PB15-PB8)
sw = GPIOB->IDR & 0x0010;    // sw = 000e0000  (mask all but bit 4)
                             // Result is sw = 00000000 or 00010000
if (sw == 0x01)              // NEVER TRUE for above sw, which is 000e0000
if (sw == 0x10)              // TRUE if e=1 (bit 4 in result of PORTB & 0x10)
if (sw == 0)                 // TRUE if e=0 in PORTB & 0x10 (sw=00000000)
if (sw != 0)                 // TRUE if e=1 in PORTB & 0x10 (sw=00010000)
GPIOB->ODR = 0x005a;         // Write to 16 bits of GPIOB; result is 01011010
GPIOB->ODR |= 0x10;          // Sets only bit e to 1 in GPIOB  (GPIOB now hgf1dcba)
GPIOB->ODR &= ~0x10;         // Resets only bit e to 0 in GPIOB  (GPIOB now hgf0dcba)
if ((GPIOB->IDR & 0x10) == 1)      // TRUE if e=1 (bit 4 of GPIOB)
```

Shift operators

Shift operators:

x >> y (right shift operand x by y bit positions)
x << y (left shift operand x by y bit positions)

Vacated bits are filled with 0's.

Shift right/left fast way to multiply/divide by power of 2

```
B = A << 3;                A   1 0 1 0 1 1 0 1
(Left shift 3 bits)        B   0 1 1 0 1 0 0 0

B = A >> 2;                A   1 0 1 1 0 1 0 1
(Right shift 2 bits)       B   0 0 1 0 1 1 0 1

B = '1';                   B = 0 0 1 1 0 0 0 1 (ASCII 0x31)
C = '5';                   C =  0 0 1 1 0 1 0 1 (ASCII 0x35)
D = (B << 4) | (C & 0x0F);
          (B << 4)       = 0 0 0 1 0 0 0 0
        (C & 0x0F)       = 0 0 0 0 0 1 0 1
                 D       = 0 0 0 1 0 1 0 1 (Packed BCD 0x15)
```

Some on-line C tutorials

- http://www.cprogramming.com/tutorial/c-tutorial.html
- http://www.physics.drexel.edu/courses/Comp_Phys/General/C_basics/
- http://www.iu.hio.no/~mark/CTutorial/CTutorial.html
- http://www2.its.strath.ac.uk/courses/c/