CprE 288 – Introduction to Embedded Systems

Instructors:
Dr. Zhao Zhang (Sections A, B, C, D, E)
Dr. Phillip Jones (Sections F, G, J)

Overview

- Announcements
- Function Calls
- Control Flow
  - for, if, else, switch, while, etc.
- Structs
- Pointers
- Lab 2

Announcements

- Homework 2 due in class on Thursday

FUNCTION CALLS

Function Calls (short intro)

- Syntax is just like Java
- Parameters can be passed by
  – value
  – address (will cover in detail after introducing pointers)

Example of calling a function:

myFunction(param1, param2);

- Implicit Declaration warning – these occur if you try to call a function that hasn't been defined yet!

Function Calls (short intro)

- All functions have
  – a return type (examples: char, void, int)
  – a name
  – a parameter list (or no parameters)
- Functions that have a return type (not void), should have a return statement

```
int add(int x, int y)
{
    return x + y;
}
```

Function Calls (short intro)

```
int add(int x, int y)
{
    return x + y;
}

void main()
{
  int r = 5;
  r = add(3, 3);
  // r is now 6
}
```


Function Calls (short intro)

```
void main()
{
  int r = 5;
  r = add(3, 3);    // Warning - implicit declaration
  // r is now 6
}

int add(int x, int y)
{
    return x + y;
}
```


Function Calls (short intro)

```
int add(int x, int y);  // best practice: add at top of file,
                        // or include a header file
void main()
{
  int r = 5;
  r = add(3, 3);
  // r is now 6
}

int add(int x, int y)
{
    return x + y;
}
```


CONTROL FLOW IN C

Reserved Words: Control Flow

- char
- double
- float
- int
- long
- short
- void

- enum
- struct
- union
- typedef

- **break**
- **case**
- **continue**
- **default**
- **do**
- **else**
- **for**
- **goto**
- **if**
- **return**
- **switch**
- **while**

- auto
- const
- extern
- register
- signed
- static
- unsigned
- volatile

- sizeof

Control Flow in C

- Control Flow – Making the program behave in a particular manner depending on the input given to the program.

- Why do we need Control Flow?
  - Not all program parts are executed all of the time, i.e., we want the program to intelligently choose what to do.

Control Flow in C

- REMEMBER! The evaluation for Boolean Control Flow is done on a TRUE / FALSE basis.
- TRUE / FALSE in the context of a computer is defined as
  - non-zero (TRUE)
  - zero (FALSE)

Examples:

-1, 5, 15, 225, 325.33	TRUE
0	FALSE

Control Flow in C: if, else if, else statement

Example

```
if (nVal > 10) {
  nVal += 5;
} else if (nVal > 5) {
  // If we reach this point, nVal must be <= 10
  nVal -= 3;
} else {
  // If we reach this point, nVal must be <= 10
  // and nVal must be <= 5
  nVal = 0;
}
```


Control Flow in C: If statement

- Must always have *if* statement; *else if* and *else* are optional

Follows a level hierarchy

- *else if* statements are only evaluated if all previous *if* and *else if* conditions have failed for the block
- *else* statements are only executed if all previous conditions have failed

Control Flow in C: comparison

Comparison (Relational Operators) – Numeric

`>, >=`

`<, <=`

`==` Equality

`!=` Not Equal

- Comparison expression gives a result of zero (FALSE) or non-zero (TRUE).
  - *A TRUE result may not necessarily be a 1*
- Equality: Double equals sign ==
  - = Assigns a value
  - == Tests for equality, returns non-zero or zero

`if (nVal == 5)` versus `if (nVal = 5)`

The second expression always evaluates to TRUE. Why?

Control Flow in C: Boolean Logic

Comparison – Multiple Conditions

Tie together using Boolean (logical) operators

`&&`	AND	& bitwise
`||`	OR	| bitwise
`!`	NOT	^ bitwise

Examples:

```
if ( (nVal > 0) && (nArea < 10))

if( (nVal < 3) || (nVal > 50))

if ( ! (nVal <= 10) )
```


Control Flow in C: Boolean Logic

A Boolean expression has a value

- A relational or logical operator produces a value of 0 or 1
- Note items in C have or produce a value: array, function, operators

What's the value of flag?

```
int nVal = 10, flag;

flag = (nVal < 0);

flag = (nVal > 0);

flag = (nVal < 3) || (nVal > 50);

flag = nVal && nVal;     // This is a tricky one
```

Control Flow in C: Boolean Logic

- WARNING!
  - Do not confuse bitwise AND, OR, and NOT operators with there Boolean counterparts

Control Flow in C: comparison

- Conditions are evaluated using *lazy evaluation*
  - Lazy evaluation – Once a condition is found that completes the condition, stop evaluating
  - OR any condition is found to be TRUE (1 OR'ed with anything = 1)
  - AND any condition is found to be FALSE (0 AND'ed with anything = 0)
- Why is lazy evaluation important?
  - Makes code run faster – skips unnecessary code. Once know condition will/will not evaluate, why evaluate other terms
- Can use lazy evaluation to guard against unwanted conditions
  - Checking for a NULL pointer before using the pointer

```
if (str && *str != '\0')
   …
```


More on conditions and testing…

Remember, conditions are evaluated on the basis of zero and non-zero.

The quantity 0x80 is non-zero and therefore TRUE.

```
if (3 || 6)
```

True or False?

Control Flow in C: Switch Statement

<u>Switch statement</u> Ex: count zeros and ones

```
switch (n) {
  case 0:
    zero_counter++;
    break;
  case 1:
    one_counter++;
    break;
  default:  // n is not equal to 0 or 1
    others_counter++;
}
```


Control Flow in C: Switch Statement

<u>Switch statement</u>

```
switch (n) {
    case 15:
    case 17:
        x = 0;
        break;
    case 32:
        x = 1;
        break;
    default:
        x = 2;
}
```

<u>Equivalent if/else if/ else</u>

```
if (n == 17 || n == 15) {
   x = 0;
} else if (n == 32) {
   x = 1;
} else {
   x = 2;
}
```


Control in C: Switch statement

- Benefit over if/else if/else
  - Compiler creates a binary tree of the cases, which reduces the number of jumps
  - Increases code readability
  - Allows falling through cases if the **break** is omitted for a case

Control Flow in C: For loop

// Syntax

for (initialization; conditional; loop) {
    / loop body */*
}

Control Flow in C: For loop

// Syntax

Note the use of semicolons

for (initialization; conditional; loop) {
    / loop body */*
}

Control Flow in C: For loop

// Best Practice

```
for (int i = 0; i < 10; i++) {
    // loop body
}
```

- The Initialization expression executes only once when first encountering the for loop.
- The Conditional expression executes at the beginning of each loop iteration; if false, control does not continue looping.
- The Loop expression execute at the end of each loop iteration.

Control Flow in C: For loop

// Equivalent loop with bad style

```
i = 0;
for (; i < 10;) {
    // loop body
    i++;
}
```


Control Flow in C: For loop

For loop

Example: calculate the sum of an array

```
for (i = 0, sum = 0; i < N; i++) {
    sum += X[i];
}
```


Control Flow in C: While loop

// Syntax

```
while (condition)  {
   // loop body
}
```

Control Flow in C: While loop

While loop

Example: calculate the length of a string

```
int strlen(char *s) {
  int n = 0;         // string length

  while (s[n]) {
    n++;
  }

  return n;
}
```


Control Flow in C: do-while loop

// Syntax

do {

// loop body

} while (condition);

Control Flow in C: do-while loop

Do-while loop

```
int i = 0, sum = 0;

do {
  sum += X[i];
} while (i++ < N);
```

- Q: What's the difference from the previous for loop?
  - A: The first iteration of the loop is always run, even if N is zero!

Control Flow in C: Break statement

Break: Exit from the immediate for, do, while, or switch statement

```
int index = -1;

// Find the index of the "Lucky" element
for (i = 0; i < N; i++) {
    if (myNumbers[i] == 7) {
        index = i;
        break;
    }
}
```

- `index` contains the index of the element equal to 7, or `index` is -1 if no element equals 7

Control Flow in C: Continue statement

Continue statement: Start the next iteration of loop

```
for (i = 0; i < N; i++) {
     /* do pre-processing for all integers */
    …

    if (X[i] < 0) {
      continue;
    }

    /* do post-processing for positives */
    …
}
```


Control Flow in C: Goto statement

- Don't use goto
  - Because Dijkstra says so
- Allows programmer to label code, then goto a spot in code using a goto label statement.

ENUM, STRUCT, UNION, TYPEDEF

Reserved Words in C

- char
- double
- float
- int
- long
- short
- void

- **enum**
- **struct**
- **union**
- **typedef**

- break
- case
- continue
- default
- do
- else
- for
- goto
- if
- return
- switch
- while

- auto
- const
- extern
- register
- signed
- static
- unsigned
- volatile

- sizeof

enum

- http://en.wikipedia.org/wiki/Enumerated_type

enum

- The enum type allow a programmer to define variable that may set to equal to a set of user defined names

```
enum compass_direction{
   north,
   east,
   south,
   west
};

enum compass_direction my_direction;
my_direction = west;
```


struct

- http://en.wikipedia.org/wiki/Struct_(C_programming_language)

struct

- The struct type allows a programmer to define a compound data type

```
struct RGB{
   char red;
   char green;
   char blue;
};

struct RGB my_color;
my_color.blue = 255;

// struct RGB *my_color_ptr = &my_color;
struct RGB *my_color_ptr = (struct RGB *) malloc(sizeof(struct RGB));

(*my_color_ptr).blue = 255;
my_color_ptr->blue = 255;          // equivalent to previous line
```

Bitfields

```
struct MyBitField{
    char clockselect : 3;
    char clockenable : 1;
    char operationmode : 4;
};
```


union

- http://en.wikipedia.org/wiki/C_language_union

union

Union: Merge multiple components

```
union u_tag {
  int ival;
  float fval;
  char *sval;
};
```

The size of a union variable is the size of its maximum component.

Structure and Union

Use of union inside of a struct

```
struct {
  char *name;
  int flags;
  int utype;
  union {
    int ival;
    float fval;
    char *sval;
  } u;
} symtab;
```


typedef

- **typedef** – a keyword used to assign alternative names to existing types
- By C coding convention, types defined with typedef should end with _t (examples: uint8_t, size_t)
- http://en.wikipedia.org/wiki/Typedef

typedef examples

```
typedef char int8_t;

typedef struct RGB{
    int8_t red;
    int8_t green;
    int8_t blue;
} RGB_t;

RGB_t my_color;
my_color.blue = 255;
```

POINTERS

Pointers

- What is a pointer?

Pointers: Mailbox Analogy

From Stoytchev's CprE 185 lecture notes

A letter fits comfortably in this box

A parcel does not. So, they give you a key …

… the key opens a larger mailbox …

... the parcel is stored there.

This is the pointer to the parcel.

Pointers

- Pointers hold the address to another variable
- You should understand these basic operations:

Operation	Mailbox Analogy
Set the pointer to the address of a variable	get the key for a certain mailbox
Dereference the pointer	get the value of the parcel
Set the value of the dereferenced object	set the value of the parcel
Increment the pointer	get the key for the next mailbox

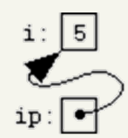
- Pointers are declared using the * character

```
int* ptrl;          // pointer to type int
int *ptr2;          // alternative declaration
char* ptr3;         // pointer to type char
int** ptr4;         // pointer to an int pointer
```


Pointers

- Setting the pointer to the address of a variable
  - & is the address operator
  - **&myVariable** is the address of **myVariable**
- Gets a mailbox address for a given parcel

```
int i = 5;
int* ip = &i;
```

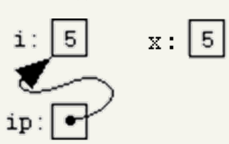


[http://www.eskimo.com/~scs/cclass/notes/sx10a.html]

Pointers

- To dereference a pointer, use the * operator before the pointer's variable name
- Gets a parcel from a given mailbox address

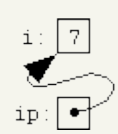
```
int i = 5;
int* ip = &i;
int x = *ip;
// x == i == 5
```



Pointers

- To set the value of i using the pointer, simply set the dereferenced pointer
- Put a parcel in a certain mailbox
- In this case, *ip = 7 is equivalent to i = 7

```
int i = 5;
int* ip = &i;
*ip = 7;
```



Pointers

- **WARNING!** A * operator is used for both dereferencing and for declaring a pointer.

```
int i = 5;
int *ip = &i; // no dereference
*ip = 7;      // dereference and assign
```

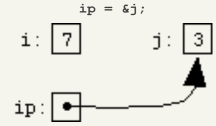
- Think of the second statement as

```
(int*) ip = &i;
```


Pointers

- Pointers can be reassigned to point to different objects
- Multiple pointers can point to the same object
- Pointers can point to memory space that exists outside your program or memory that doesn't exist (causes an error)

```
int i = 5;
int* ip = &i;
*ip = 7;
int j = 3;
ip = &j;
```



Pointers

- Incrementing and decrementing a pointer
  - Increments/decrements by the size of the type
- Example (on a byte addressed system)
  - int* increment by 2 (int's are 2 bytes on the ATmega 128)
  - char* increment by 1

```
int* ip = 1000;          // sizeof(int) == 2
char* cp = 1000;         // sizeof(char) == 1
ip++;
cp++;
// ip == 1002 and cp = 1001
```


Pointers

- Pointers are useful for passing parameters to a function by reference (instead of value)
  - Especially useful when the variables consume lots of memory
  - Java Objects use the same concept of pointers, as Objects are passed to functions by reference

Pass by Reference Example

```
void addThree(int *ptr) {
    *ptr += 3;
}

void main() {
   int x = 5;
   addThree(&x);
   // x is now 8
}
```


Pointer Example

char s = 5;	
char t = 8;	
char *p1 = &s;	• p1 points to s
char **p2 = &p1;	• p2 points to p1
*p1 = 9;	• Same as: ***s = 9;***
**p2 = 7;	• Same as: ****p1 = 7;*** or ***s = 7;***
*p2 = &t;	• Same as: ***p1 = &t;*** (p1 now points to t)
*p1 = 10;	• Same as: ***t = 10;***

Pointer Example

```
char r = 10;          *p1 = 20;     s = 20;
char s = 15;          *p2 = 30;     t = 30;
char t = 13;          **p3 = 40;    s = 40;
char *p1 = &s;        *p3 = &t;     p1 = &t;
char *p2 = &t;        **p3 = 50;    t = 50;
char **p3 = &p1;
                      p3 = &p2;
                      *p3 = &r;     p2 = &r;
```


Exercise: Pointer

char msg[] = “Welcome to CprE 288”;
char *str;

Which of the following statements are good (valid and serve the purpose)?

a. str = msg[0];
b. str = msg;
c. str = &msg[10];
c. *str = msg;
d. *str = &msg[0];
e. *str = msg[10];

Exercise: Pointer

Assume the AVR platform, the address of x is 0x0200, the address of y is 0x0202.

```
int x = 100, y = 200;
int* p1 = &x;
int* p2 = &y;
*p2 = *(p1++);
```

At the end

x = ________
y = ________
p1 = ________
p2 = ________

Exercise: Pointer, Array and Function

int len;
char msg[] = “Microcontrollers are tons of fun!”;

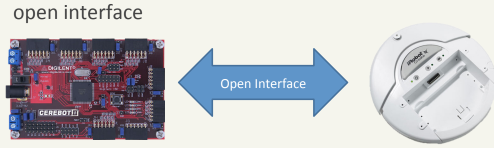
Write a loop to calculate the length of *msg* and put it into *len*

a. Use pointer access
b. Use array access

LAB 2 OVERVIEW

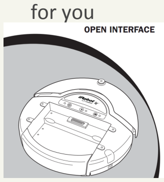
Open Interface

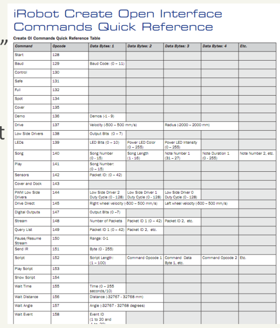
- Program is on the MCU (ATmega128 processor)
- Motors for movement are on the iRobot
- Communication occurs over a standard RS232 serial port using UART0
- This communication has been abstracted by using the open interface



Open Interface

- Open Interface makes it so you don't have to "see" the serial communication
- You simply call functions that handle the serial part for you





iRobot Create Open Interface Commands Quick Reference

Create OI Commands Quick Reference Table

Command	*Opcode*	*Data Bytes: 1*	*Data Bytes: 2*	*Data Bytes: 3*	*Data Bytes: 4*	*Etc.*
Start	128					
Baud	129	Baud Code: (0 – 11)				
Control	130					
Safe	131					
Full	132					
Spot	134					
Cover	135					
Demo	136	Demos (-1 - 9)				
Drive	137	Velocity (-500 – 500 mm/s)		Radius (-2000 – 2000 mm)		
Low Side Drivers	138	Output Bits (0 – 7)				
LEDs	139	LED Bits (0 – 10)	Power LED Color (0 – 255)	Power LED Intensity (0 – 255)		
Song	140	Song Number (0 - 15)	Song Length (1 - 16)	Note Number 1 (31 – 27)	Note Duration 1 (0 - 255)	Note Number 2, etc.
Play	141	Song Number: (0 – 15)				
Sensors	142	Packet ID: (0 – 42)				
Cover and Dock	143					
PWM Low Side Drivers	144	Low Side Driver 2 Duty Cycle (0 - 128)	Low Side Driver 1 Duty Cycle (0 - 128)	Low Side Driver 0 Duty Cycle (0 - 128)		
Drive Direct	145	Right wheel velocity (-500 – 500 mm/s)		Left wheel velocity (-500 – 500 mm/s)		
Digital Outputs	147	Output Bits (0 –7)				
Stream	148	Number of Packets	Packet ID 1 (0 – 42)	Packet ID 2, etc.		
Query List	149	Packet ID 1 (0 – 42)	Packet ID 2, etc.			
Pause/Resume Stream	150	Range: 0-1				
Send IR	151	Byte (0 - 255)				
Script	152	Script Length: (1 – 100)	Command Opcode 1	Command Data Byte 1, etc.	Command Opcode 2	Etc.
Play Script	153					
Show Script	154					
Wait Time	155	Time (0 – 255 seconds/10)				
Wait Distance	156	Distance (-32767 - 32768 mm)				
Wait Angle	157	Angle (-32767 - 32768 degrees)				
Wait Event	158	Event ID (1 to 20 and				

Open Interface

```
// Allocate a sensor struct
oi_t* oi_alloc();

// Initialize the serial communication
void oi_init(oi_t *self);

// Update the oi_t sensor struct
void oi_update(oi_t *self);

// Set velocity of each wheel in mm/s (value should be between -500 and +500)
void oi_set_wheels(int16_t right_wheel, int16_t left_wheel);
```


Open Interface

- Initializing the serial connection

```
// Make sure the iRobot is powered on
oi_t* sensor_status = oi_alloc();     // allocate memory
oi_init(sensor_status);               // initialize
```


Open Interface

- oi_t* sensor_status
  - it's a struct for keeping the state of the iRobot
  - necessary since the status of sensors can only be current if serial communication is used
  - call **oi_update(sensor_status);** to refresh the members of the struct

```
typedef struct {
        // Boolean value for the right bumper
        uint8_t bumper_right;
        // Boolean value for the left bumper
        uint8_t bumper_left;
        // Boolean value for the right wheel
        uint8_t wheeldrop_right;
        // Boolean value for the left wheel
        uint8_t wheeldrop_left;

        // ... a lot more variables
} oi_t;
```


Move the Robot Forward

```
#include "open_interface.h"
#include "util.h"

void main() {
  oi_t *robot = oi_alloc();
  oi_init(robot);

  oi_set_wheels(250, 250);
  wait_ms(5000);
  oi_set_wheels(0, 0);

  free(robot);
}
```


Move Forward

```
#include "open_interface.h"
#include "util.h"

int moveForward(oi_t *self, unsigned int distance_mm) {
  oi_set_wheels(250, 250);
  int sum = 0;
  while (sum < distance_mm) {
    oi_update(self);
    sum += self->distance;
    // optional check for bump sensors
  }
  oi_set_wheels(0, 0);

  return sum;
}

void main() {
  oi_t *robot = oi_alloc();
  oi_init(robot);

  moveForward(robot, 1000);

  free(robot);
}
```

iRobot Open Interface and Movement

Lab 2, Part II. Robots moving in a square

New functions involved:

```
// return current angle in degree
int oi_current_angle(oi_t *self) ;
// reset current record of angle
void oi_clear_angle(oi_t *self);
```


iRobot Open Interface and Movement

Lab 2, Part III. Bump detection

New function involved:

```
//Returns bump sensor status
// 0 = no sensors pressed
// 1 = right sensor
// 2 = left sensor
// 3 = both sensors
char oi_bump_status(oi_t *self);
```


iRobot Open Interface and Movement

What you will learn:

- How to program robot behavior using a set of API functions
- How API functions simplifies a programmer's job

Common approaches when working with I/O devices