# Autonomous Mobile Robots

## Lecture 01: Introduction

Lecture is based on material from Robotic Explorations: A Hands-on Introduction to Engineering, Fred Martin, Prentice Hall, 2001.

## Acknowledgement

This collection of eight lectures was prepared for the Autumn 2000 EE462 class, “Principles of Mobile Robots,” at the University of Washington using Fred Martin’s pre-publication text, Robotic Explorations: A Hands-on Introduction to Engineering, Prentice Hall, 2001.

The PowerPoint slides were created by Dr. Linda Bushnell, bushnell@ee.washington.edu

Please see the EE462 course web site for more information on the syllabus, laboratory assignments, homework assignments, and links: http://www.ee.washington.edu/class/462/bushnell/

## Outline

- Introduction to the Course
- The Technology
- The Laboratory
- Lab Assignment #1 - building the HandyBug
- Braitenberg Vehicles
- The Interactive C Language
- MIT 6.270 Autonomous Robot Design Competition videos

## Homework #1

- **Motivation for Class:** Read Chapter 1 of Robotic Explorations (textbook)
- **Interactive C:** Read Appendix E of Robotic Explorations (textbook) and pp. 1-37 of The Handy Board Reference Manual.
- **A First Robot:** Read Chapter 2 of Robotic Explorations (textbook)

## Introduction

- **Course Description:**
  - This course uses LEGO beams, plates, gears, motors, the Handy Board microcontroller board programmed in Interactive C and various sensors to construct autonomous mobile robots.
  - The first half of the course contains four structured laboratory exercises in LEGO mechanics, software design, sensor and motor principles and control.
  - The second half of the course laboratory is spent designing mobile robots that can compete in a competition.
  - The lectures will focus on IC, the Handy Board, motors, sensors, and various control methodologies.

## Introduction

- **Goals:**
  - Integrated design (mechanics, electronics, software)
  - Control systems (PID vs. Algorithmic vs. Reactive)
  - Interdisciplinary teamwork and problem-solving
- **Topics:**
  - Programming the Handy Board using Interactive C
  - Mechanical construction using LEGOs
  - Motors
  - Sensors and Advanced Sensing
  - PID control, Algorithmic control, Reactive control

## Introduction

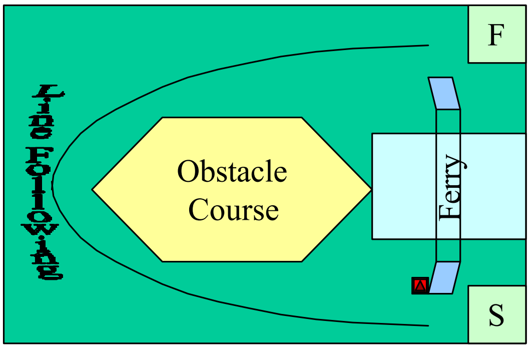
- **Textbook:**
  - Robotic Explorations: A Hands-on Introduction to Engineering, Fred Martin, Prentice Hall, 2001.
- **Reference Textbook:**
  - J. L. Jones, A. M. Flynn and B. A. Seiger, Mobile Robots: Inspiration to Implementation, A.K. Peters, 2nd Edition, 1999.
- **Other Documents:**
  - Handy Board Reference Manual
  - Interactive C Manual (Chapter 5 of HB Reference Manual)
  - “The Art of LEGO Design,” by Fred Martin
  - 6.270 Hardware Reference Manual (from MIT LEGO Robot course)
  - M68HC11 Reference Manual (Motorola Microprocessor)
  - Various papers to be assigned for reading

## Introduction

- **Homework:**
  - There will be 8 homeworks based on the lecture material.
- **Lab Assignments:**
  - The first six weeks will have structured laboratory assignments:
    - Week 1 & 2: Form teams of 2-3; Construct LEGO HandyBug and program using Interactive C
    - Week 3: Motors
    - Week 4: Sensors
    - Week 5 & 6: PID Control, polarized light sensing, data collection
  - A lab report will be due after the lab is completed.
  - The remainder will be unstructured lab time where you will design, build and test a mobile robot for the competition.

# Introduction

- **The Competition:**
  - Each team will design, construct and test a mobile robot that will be competed at the end of the class (date TBD).
  - Maze with three paths, each requiring different robot features:
    - Line following
    - Obstacle course
    - Ferry crossing



# The Technology

- Hardware: Handy Board - hand-held microprocessor-based robot control board. Ideal for controlling small, mobile robots.

- Software: Interactive C - custom software environment for the Handy Board.

- Mechanism: LEGO Technic system - extension of LEGO building brick system for constructing mobile robot.

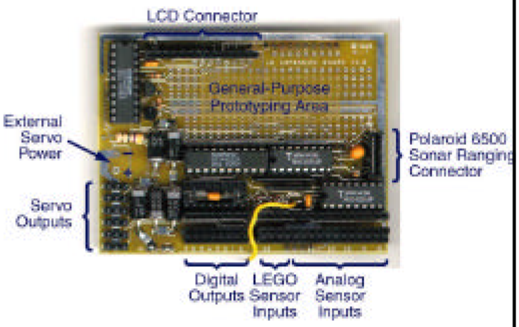
## The Technology

- **The Handy Board:**
  - Motorola 68HC11 8-bit microprocessor
  - 32K main system memory, battery-protected (allows the use of Interactive C)
  - Output drivers for 4 DC motors (9v, 1A)
  - Inputs for analog and digital sensors; up to 7 analog sensors and 9 digital sensors
  - Internal, rechargeable battery pack (in case)
  - LCD screen (16 character, 2-line liquid crystal display screen)

Easy to attach motors and sensors

## The Technology

- **The Expansion Board:**
  - 10 additional analog sensor inputs
  - 4 inputs for active LEGO sensors (reflectance sensor and shaft encoder)
  - 9 digital outputs
  - 6 servo motor control signals with power supply from the Handy Board's internal battery
  - optional external power for servo motors



– connector mount for Polaroid 6500 ultrasonic ranging system

– general-purpose electrical prototyping area

– pass-through connector for the Handy Board's LCD screen

## The Technology

- **Interactive C:**
  - C-language compiler developed for robotic applications (by Randy Sargent)
  - Interactivity - command-line console that allows user to type expressions and function calls interactively, even when other programs are running - easier to try out ideas
  - Stability - IC reports a runtime error for common programming problems (divide-by-zero, out-of-bounds array reference, etc.) rather than crashing the system
  - Multi-tasking - multiple programs running simultaneously (up to 12)

Alternatives: program Handy Board directly in 68HC11 machine language

## The Technology

- **LEGO Technic:**
  - *Technic* is LEGO's brand name for the mechanized portion of the product line
  - Includes: beams, bricks, axles, gears, motors and other parts for construction of complex and functional mechanical systems
  - No need to learn tools in a machine shop
  - Use cross-beams to lock design in place

Alternatives: Fischertechnik (more rigid, industrial), Modified RC cars, Scrap materials

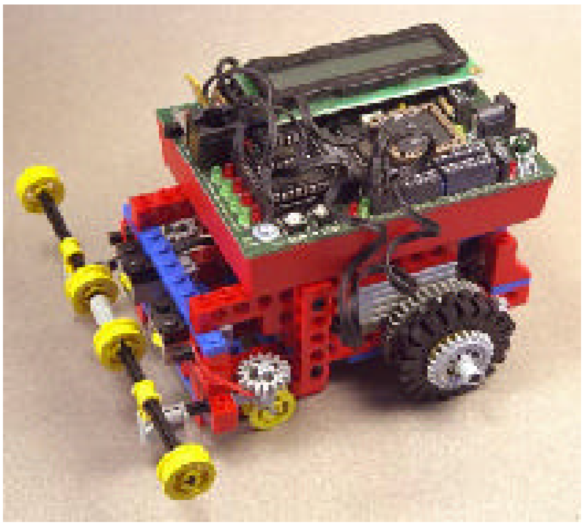
## The Laboratory

- **Robot Station:**
  - PC with serial port
  - Handy Board controller
  - Serial interface and battery charger unit for Handy Board
  - Phone cable (RJ11 modular cable) for connecting PC to Handy Board
  - Power adapter for Handy Board
  - Handy Board Technical Manual (on-line)
  - Expansion Board
  - Sensor/Motor kit
  - Interactive C
  - LEGO *Technic* Resource Set
  - Gear-reduction, servo and micro motors

## Lab Assignment #1

**The HandyBug**

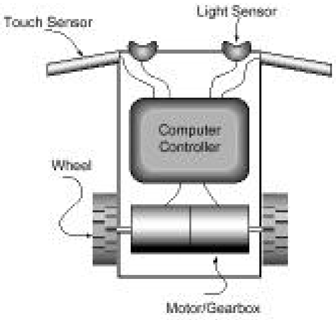
- Build the HandyBug
- Program with IC



## Lab Assignment #1

### The HandyBug

- **Building the HandyBug:**
  - 2 motors (9v LEGO motors)
  - 2 sensors (touch sensors)
  - Carries Handy Board
  - Plans given in lab handout
  - Turtle configuration: separate left-side and right-side motor drives



Schematic of LEGO Turtle Robot

## Lab Assignment #1

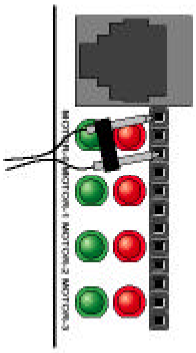
### The HandyBug

- **Interactive C:**
  - **C> beep();**
    - IC compiles line of code
    - compiled code is downloaded via serial line to the Handy Board
    - IC tells Handy Board to execute the code it has just received
    - Handy Board beeps
  - Valid C statement:
    - function call (“beep”)
    - parentheses contain arguments (parameters) to the function call
    - trailing semicolon is end-of-statement marker
  - Arithmetic expressions:
    - **C> 2 + 2;** (executed by the Handy Board)
  - Multiple statements on one line:
    - **C> {beep(); sleep(2.0); beep();}**
  - Command to IC:
    - **C> load test.c**

# Lab Assignment #1

## The HandyBug

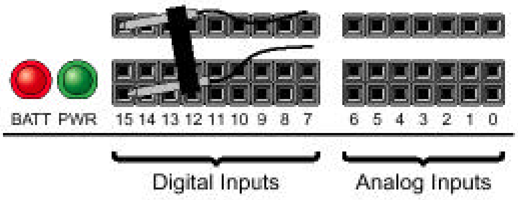
- **Motors:**
  - IC has library files for controlling the motors and sensors of the Handy Board
  - Functions:
    - **fd(0);** - Motor 0 output port turns on, green LED on, motor spins
    - **bk(0);** - motor spins in opposite direction
    - **off(0);** - motor turns off
    - **motor(0,50);** - turns motor port 0 on in the “fd” direction with a power level of 50% (port can be 0, 1, 2, 3) (power level ranges from -100 to +100; 0 is off, +100 is full on in the “fd” direction)
  - Handy Board uses Pulse Width Modulation (PWM) to control the motors (turns on and off very quickly)



# Lab Assignment #1

## The HandyBug

- **Sensors:**
  - Handy Board has inputs for 9 digital (switch-type) sensors and 7 analog (continuously varying) sensors.
  - Digital Sensors:
    - Digital inputs # 7 to 15
    - Test switch using **digital(port#);**
      - **digital(15);** Handy Board returns True/1/switch closed or False/0/switch open.
      - **if (digital(15)) {beep();}** Tests the state of the sensor.



# Lab Assignment #1

## The HandyBug

- **Files and Functions:**
  - keyword **void** - indicates function has no return value
  - **test**: function name
  - { definition of function }
  - **sleep** - creates a delay
  - semicolon after each statement
  - save as **test.c** in IC folder
  - **C> load test.c** (no semicolon)
  - **C>test();** (runs program)
  - **Result:** motor 0 port turns on in forward direction for 1 second, switches to backward direction for 1 second, turns off, beeps.

**test.c**

```
void test () {
    fd(0);
    sleep(1.0);
    bk(0);
    sleep(1.0);
    off(0);
    beep();
}
```

# Lab Assignment #1

## The HandyBug

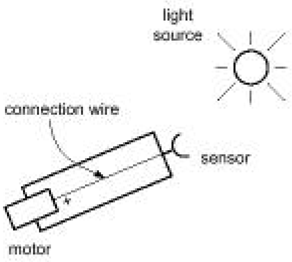
- **Main Function:**
  - **while(1) -** infinite loop
  - **/*…*/** for comments
  - **printf -** formatted print on HB
  - **\n -** newline before printing next
  - **main()** - HB automatically loads when turned on
  - to reset HB without running main(), hold down START button when turning on HB

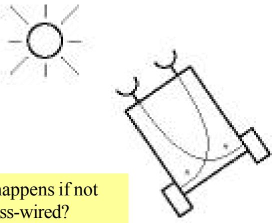
**robot.c**

```
/* sample robot program */
void main() {
    while (1) {
            printf(“Going forward…\n”);
            fd(0);
            if (digital(15)) {
                    printf(“Backing up…\n”);
                    bk(0);
                    beep();
                    sleep(2.0);
            }
    }
}
```

# Braitenberg Vehicles

- Neuro-biologist Valentino Braitenberg, *Vehicles: Experiments into Synthetic Psychology* (1984). “how sentient creatures might have evolved from simpler organisms”
- Vehicle 1: 1 Motor/1 Sensor
  - Wire connects sensor to motor
  - Sensor generates a signal proportional to the strength of light
  - When it “sees” a light source, it starts moving in straight line
- Vehicle 2b: 2 Motors/2 Sensors
  - Turns towards light source
  - Reduces difference between heading and brightest source of light (negative feedback)



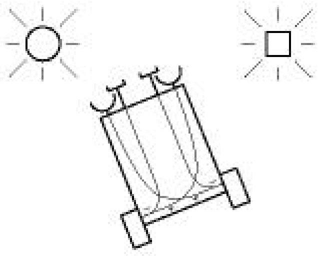


What happens if not cross-wired?

# Braitenberg Vehicles

- See web sites:
  - http://pikas.inf.tu-dresden.de/compulog/lectures/winter99/lpisa/mod_1_12.html

  for more information
  - http://people.cs.uchicago.edu/~wiseman/vehicles/ for simulations



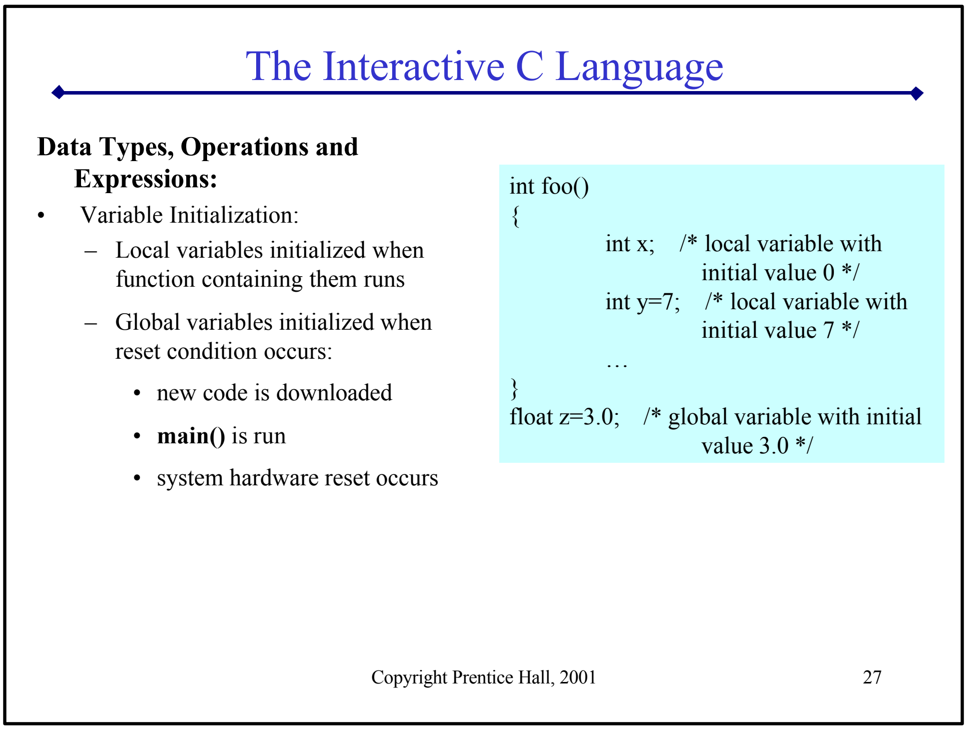
## The Interactive C Language

- C Language consisting of a compiler and a run-time machine language module
  - compiler has interactive command-line compilation and debugging
  - user's C code is converted into instructions for a specially-designed virtual machine; Handy Board is programmed to interpret these instructions
  - drawback to virtual machine approach:  execution speed
- **IC Commands:**
  - compile and load file: **load <filename>** (HB must be attached for this to work)
  - unload file: **unload <filename>**
  - list files, functions or globals: **list files**, **list functions**, **list globals**
  - kill all processes: **kill_all**
  - print process status: **ps**
  - **help, quit**

## The Interactive C Language

**Data Types, Operations and Expressions:**

- Variable names: case sensitive, use __ for readability
- Data Types supported by IC:
  - 16-bit integer - **int** - signed integers from -32,768 to +32,767
  - 32-bit integer - **long** - signed integers from -2,147,483,648 to +2,147,483,647
  - 32-bit floating point number - **float** - seven decimal digits of precision from $10^{-38}$ to $10^{38}$
  - 8-bit characters - **char** - printable symbol using standard ASCII character code
- Local and Global Variables:
  - Local: variable is declared within a function, or as an argument to a function
  - Global: variable is declared outside of a function, for all functions

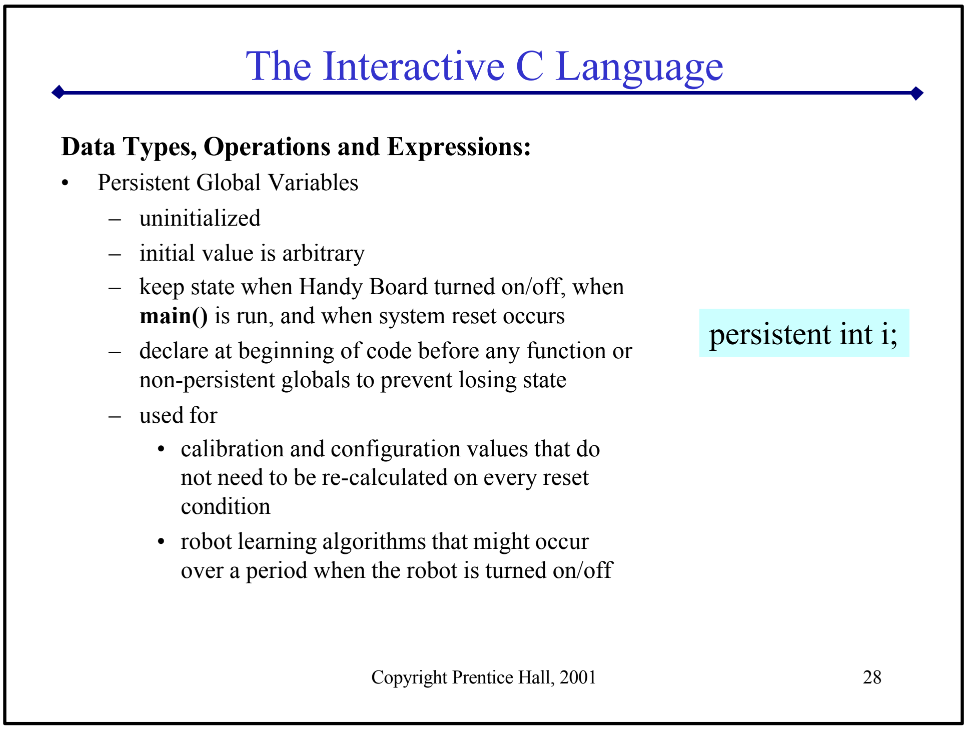


## The Interactive C Language

**Data Types, Operations and Expressions:**

- Variable Initialization:
  - Local variables initialized when function containing them runs
  - Global variables initialized when reset condition occurs:
    - new code is downloaded
    - **main()** is run
    - system hardware reset occurs

```
int foo()
{
        int x;    /* local variable with
                     initial value 0 */
        int y=7;   /* local variable with
                     initial value 7 */
        …
}
float z=3.0;   /* global variable with initial
                     value 3.0 */
```

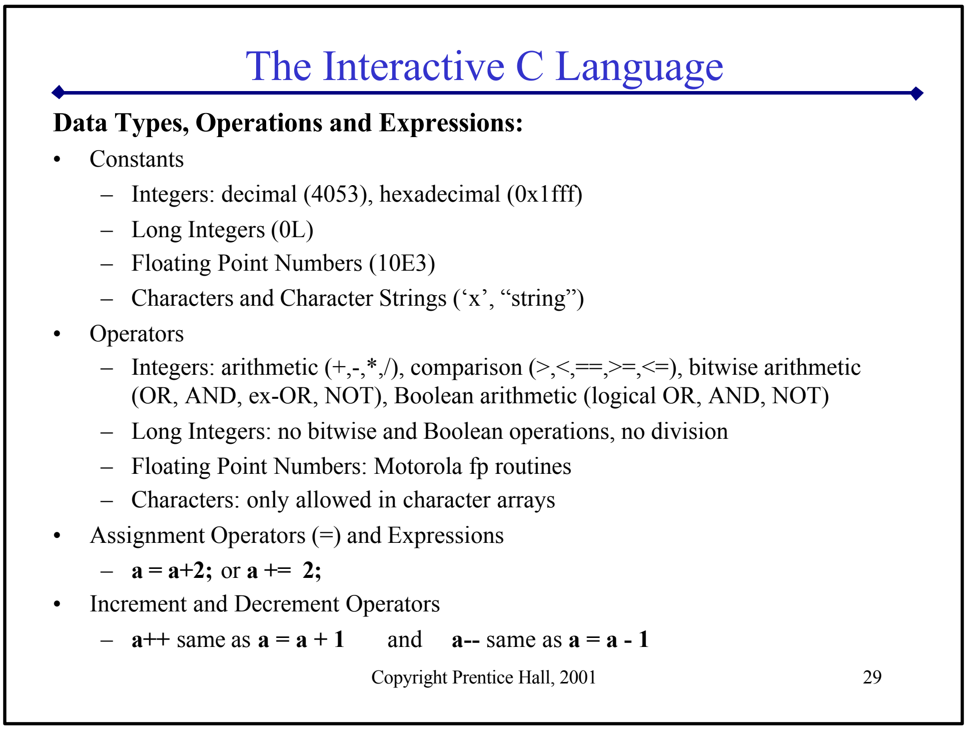


## The Interactive C Language

**Data Types, Operations and Expressions:**

- Persistent Global Variables
  - uninitialized
  - initial value is arbitrary
  - keep state when Handy Board turned on/off, when **main()** is run, and when system reset occurs
  - declare at beginning of code before any function or non-persistent globals to prevent losing state
  - used for
    - calibration and configuration values that do not need to be re-calculated on every reset condition
    - robot learning algorithms that might occur over a period when the robot is turned on/off

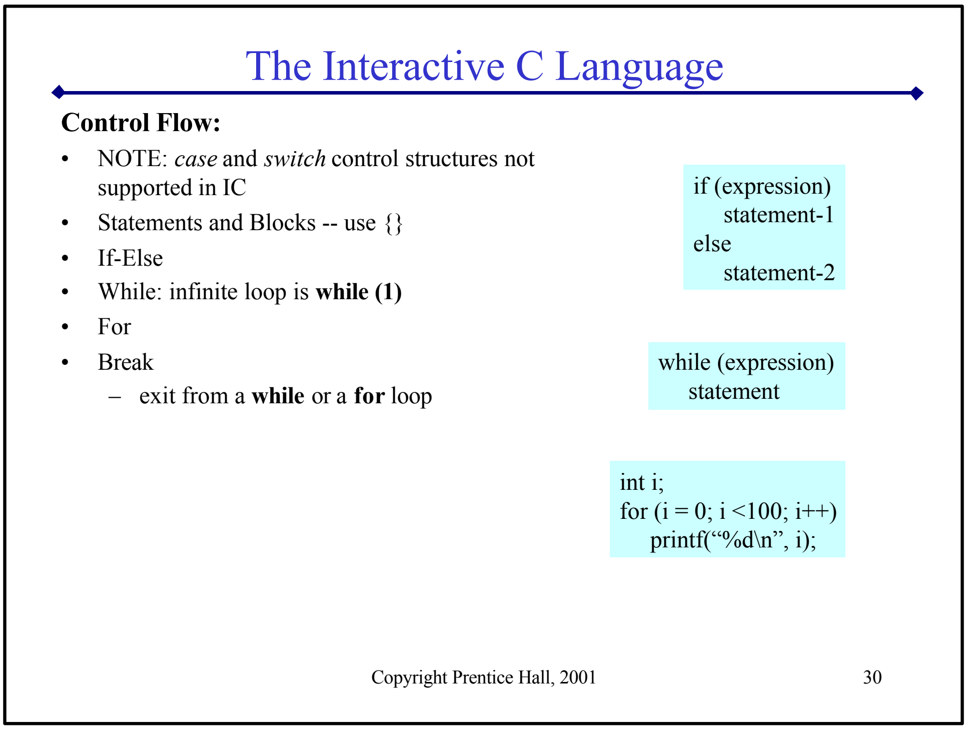
```
persistent int i;
```



## The Interactive C Language

**Data Types, Operations and Expressions:**

- Constants
  - Integers: decimal (4053), hexadecimal (0x1fff)
  - Long Integers (0L)
  - Floating Point Numbers (10E3)
  - Characters and Character Strings (‘x’, “string”)
- Operators
  - Integers: arithmetic (+,-,*,/), comparison (>,<,==,>=,<=), bitwise arithmetic (OR, AND, ex-OR, NOT), Boolean arithmetic (logical OR, AND, NOT)
  - Long Integers: no bitwise and Boolean operations, no division
  - Floating Point Numbers: Motorola fp routines
  - Characters: only allowed in character arrays
- Assignment Operators (=) and Expressions
  - **a = a+2;** or **a += 2;**
- Increment and Decrement Operators
  - **a++** same as **a = a + 1** and **a--** same as **a = a - 1**



## The Interactive C Language

**Control Flow:**

- NOTE: *case* and *switch* control structures not supported in IC
- Statements and Blocks -- use {}
- If-Else
- While: infinite loop is **while (1)**
- For
- Break
  - exit from a **while** or a **for** loop

```
if (expression)
    statement-1
else
    statement-2
```

```
while (expression)
    statement
```

```
int i;
for (i = 0; i <100; i++)
    printf(“%d\n”, i);
```

## The Interactive C Language

**Printing on LCD Screen:**

- a message
  - **\n** is end of line

```
Printf("Hello, world!\n");
```

- a number
  - **%d** is for decimal format

```
Printf("Value is %d\n", x);
```

- a number in binary
  - **%b** is for binary format
  - low byte of number is printed

```
Printf("Value is %b\n", x);
```

- a floating point number
  - **%f** is for floating point format

```
Printf("Value is %f\n", x);
```

- two numbers in hexadecimal format
  - **%x** is for hexadecimal format

```
Printf("A=%x B=%x\n", a, b);
```

- NOTE:
  - final character position on LCD screen is used as the system "heartbeat" - continuously blinks when ok
  - printf() treats 2-line LCD screen as one long line
  - no support of printing long integers

## The Interactive C Language

**Arrays (1D only) and Pointers:**

Declaring and Initializing Arrays

```
int foo[10];
```

```
int foo[] = {0, 4, 5, -8, 17, 301};
```

```
char string[] = "Hello there";
```

Declaring Pointer Variables

```
int *foo;
int x = 5;
int y;
foo = &x;
y = *foo;
```

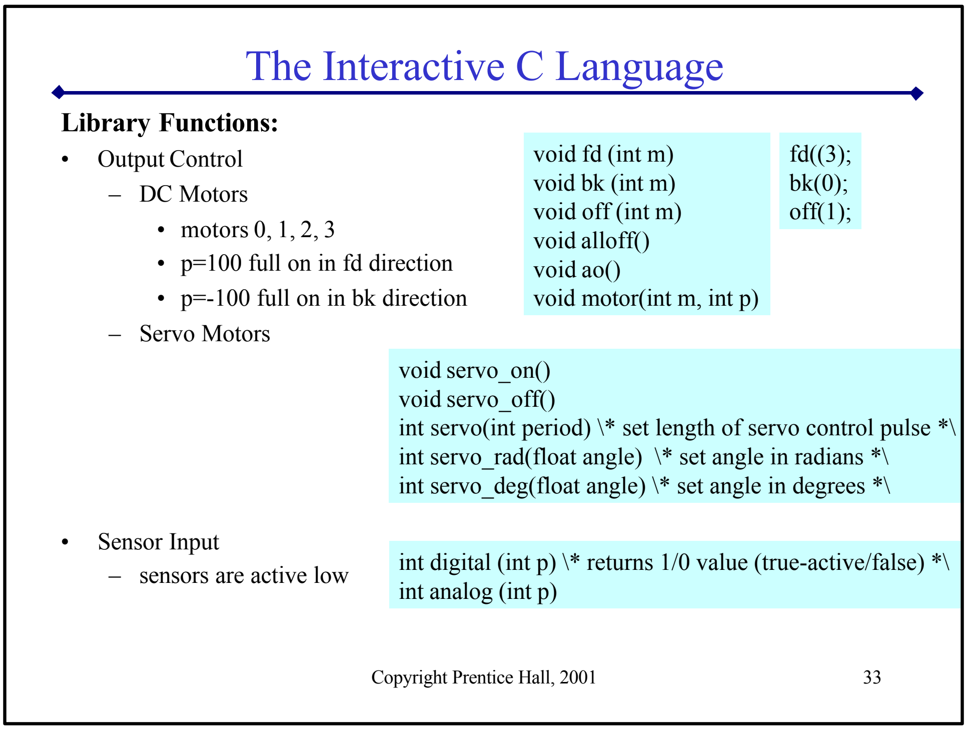
Passing Arrays as Arguments

```
int retrieve_element (int index, int array[])
{
   return array[index];
}
```

```
{
int array[10];
retrieve_element(3,array);
}
```

Passing Pointers as Arguments

```
void avg_sensor (int port, int *result)
{
   int sum = 0;
   int i;
   for (i = 0, i < 10, i ++) sum += analog(port);
   *result = sum/10;
}
```



## The Interactive C Language

**Library Functions:**

- Output Control
  - DC Motors
    - motors 0, 1, 2, 3
    - p=100 full on in fd direction
    - p=-100 full on in bk direction
  - Servo Motors

```
void fd (int m)
void bk (int m)
void off (int m)
void alloff()
void ao()
void motor(int m, int p)
```

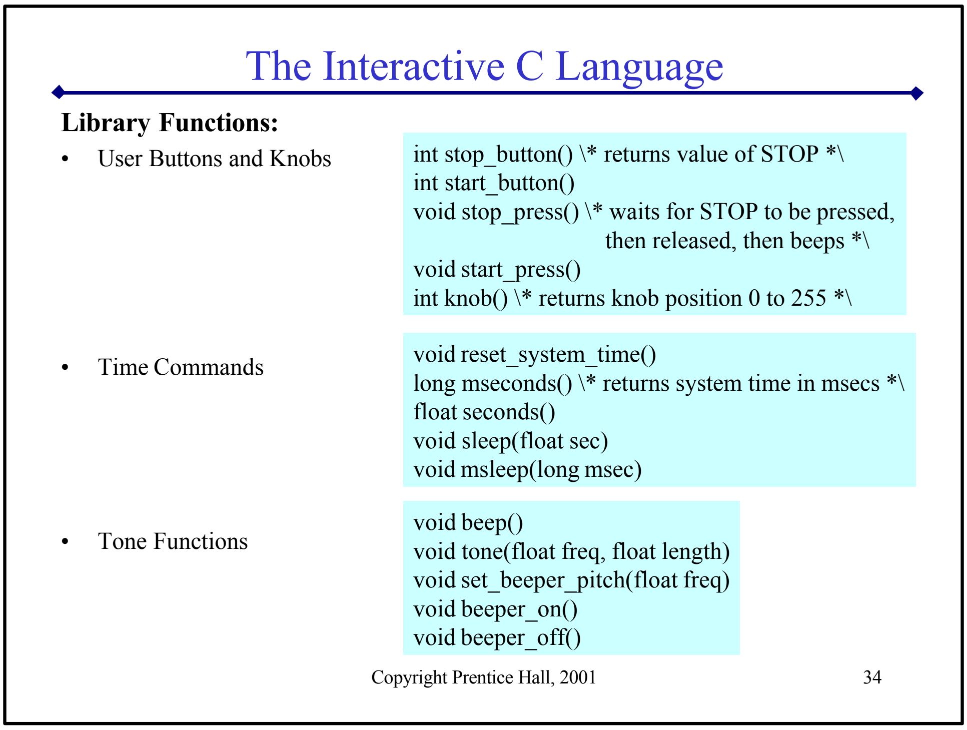
```
fd((3);
bk(0);
off(1);
```

```
void servo_on()
void servo_off()
int servo(int period) \* set length of servo control pulse *\
int servo_rad(float angle)  \* set angle in radians *\
int servo_deg(float angle) \* set angle in degrees *\
```

- Sensor Input
  - sensors are active low

```
int digital (int p) \* returns 1/0 value (true-active/false) *\
int analog (int p)
```

33



## The Interactive C Language

**Library Functions:**

- User Buttons and Knobs

```
int stop_button() \* returns value of STOP *\
int start_button()
void stop_press() \* waits for STOP to be pressed,
                          then released, then beeps *\
void start_press()
int knob() \* returns knob position 0 to 255 *\
```

- Time Commands

```
void reset_system_time()
long mseconds() \* returns system time in msecs *\
float seconds()
void sleep(float sec)
void msleep(long msec)
```

- Tone Functions

```
void beep()
void tone(float freq, float length)
void set_beeper_pitch(float freq)
void beeper_on()
void beeper_off()
```

34

## The Interactive C Language

**Multi-Tasking:**

- – Processes communicate through global variables
- – Each process runs for a certain number of *ticks, and* has its own program stack

- Creating New Processes

```
int start_process( function-call(…), [ticks], [stack-size] )
```

```
void check_sensor(int n)
{
   while (1)
            printf("Sensor %d is %d\n", n, digital(n));
void main()
{ start_process(check_sensor(2), 1, 50);} \* runs 1 ms with
                                                    stack size 50 *\
```

- Destroying Processes

```
Void main()
{
    int pid;
    pid=start_process(check_sensor(2));
    sleep(1.0);
    kill_process(pid);
}
```

- Process Management

```
C>kill_all
C>ps
```

## Video

MIT 6.270 Autonomous Robot Design Competition Videos:

- **2000 "Bots in Blue"** - Robo-CPs compete against each other to collect unwanted hackers and throw them in the brig.
- **1999 "Raiders of the Lost Parts"** - Roboarcheologists race to explore the alien ruins and retrieve valuable artifacts.
- **1998 "RoboGolf"** - Robotic golfers compete to become champion of a post-apocalyptic world.

Available for purchase at: http://web.mit.edu/6.270/www/about/video.html