CprE 288 – Introduction to Embedded Systems

Instructors

Swamy Ponpandi

Announcements

- **Final Project: Have project groups formed by Friday (Mar 24, 2017), 4.00 PM**
  - Each Final Project teams will be composed of **two** regular Lab teams combined.
  - Give or E-mail your lab section TA, the following,

    a) a list of your team members.

    b) a creative team name (*be mindful of university policies*).

Announcement

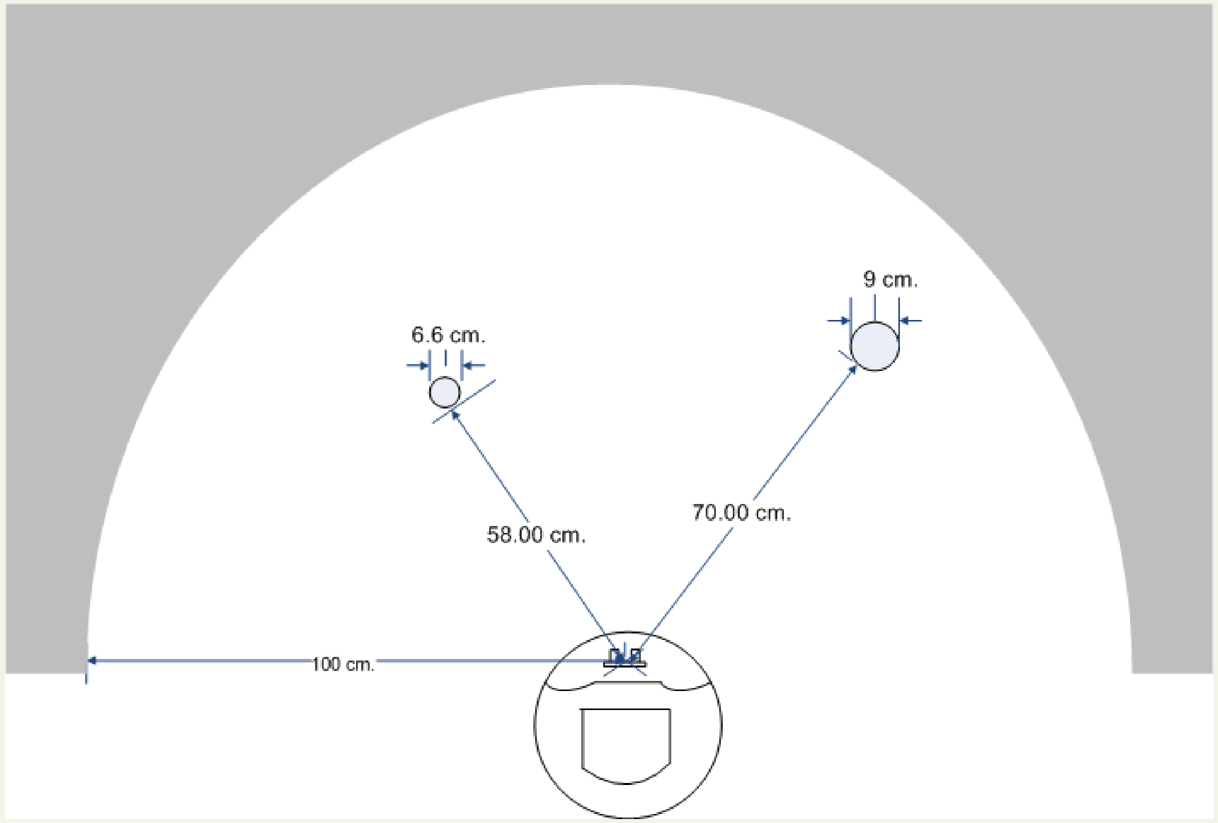
- Lab 9: Object Detection – 2 week lab

Lecture Overview

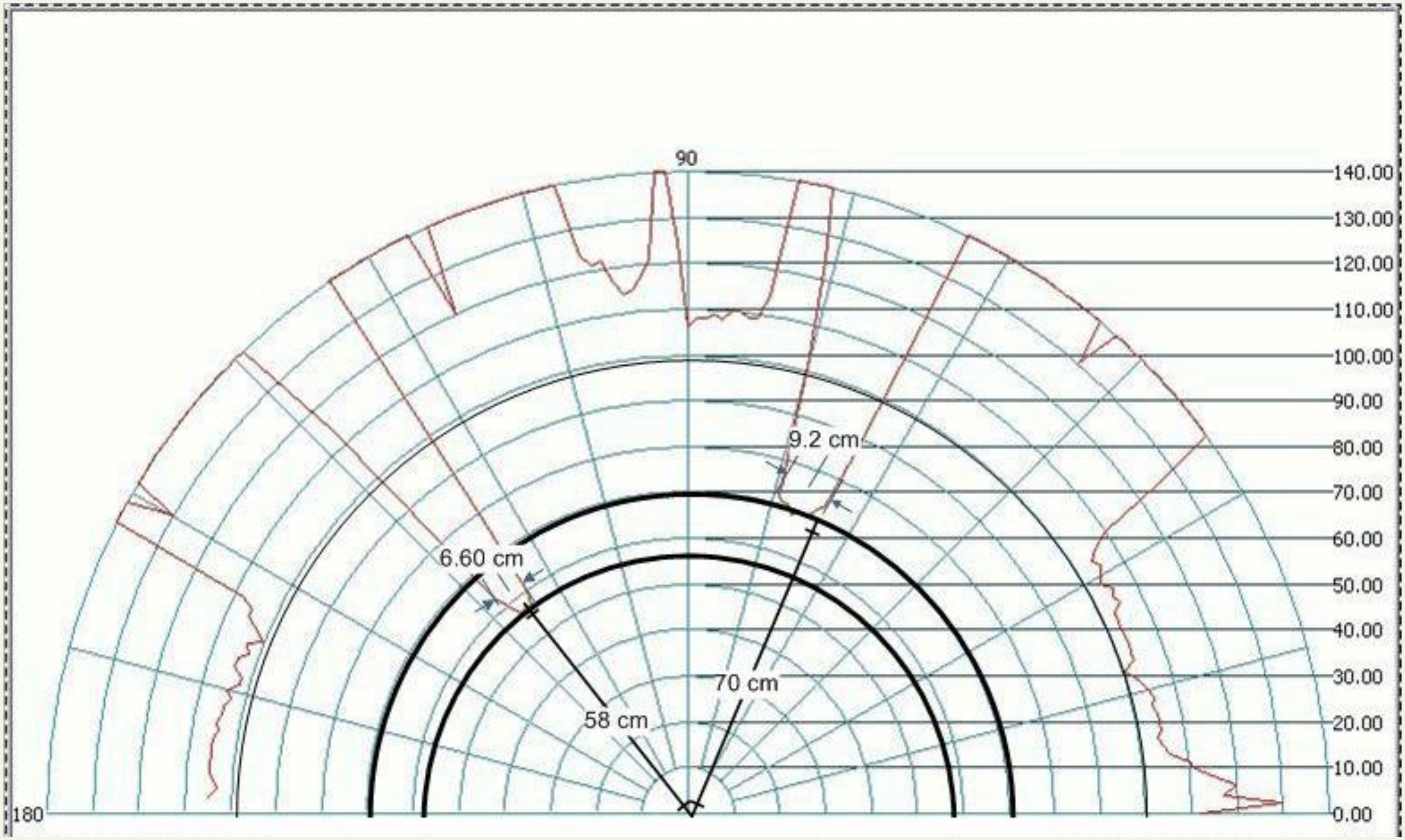
- Suggested Programming Style for Lab Project

Lab 9: Object Counting and Size Discrimination

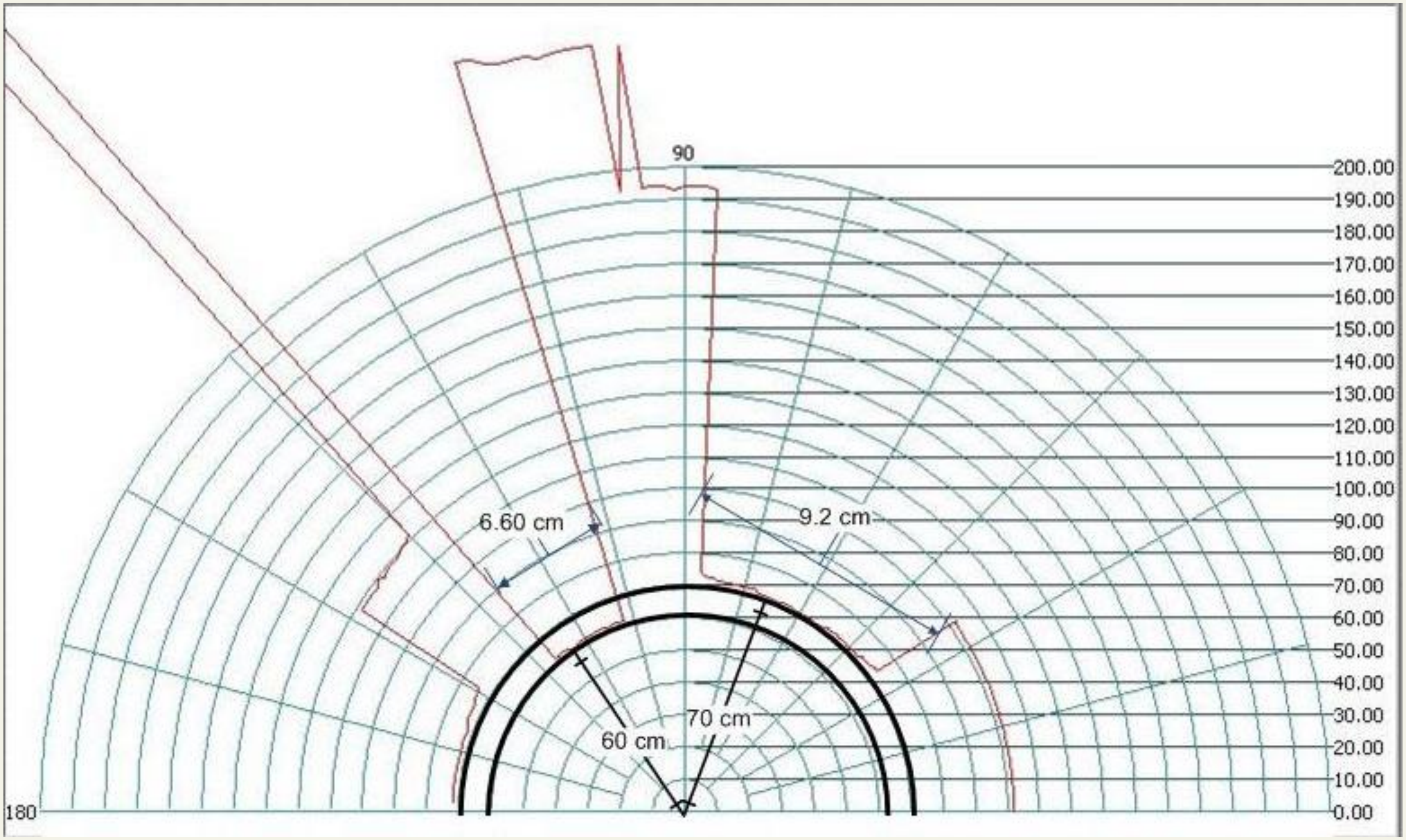
How do you distinguish two objects of different size?



Scanned Results by IR Censor



Scanned Result by Ping))) Sensor



Data Analysis

How can your program identify and distinguish different objects from the following raw data?

Degrees	IR Distance (cm)	Sonar Distance (cm)
0	120	324
2	123	330
4	119	363
6	40	40
8	40	40
10	40	41
… (more)		

Data Analysis

Step 1: Scan the array to identify gaps, convert them to **angular sizes**

- What's your algorithm?

Step 2: For each object, convert its **distance** and **angular size** into **linear size** (width)

- What's your mathematic formula?

Suggested Programming Style for Lab Project

References and Readings

- GNU Coding Standards. Free Software Foundation
- Proper Linux Kernel Coding Style. Greg Kroah-Hartman, Linux Journal, July 01, 2002
- Recommended C Style and Coding Standards. L. W. Cannon et al.
- Indent Style, Wikipedia

Credit: Jafar M. Al-Kofahi made contribution to an early version of 288 Lab Project Coding Style

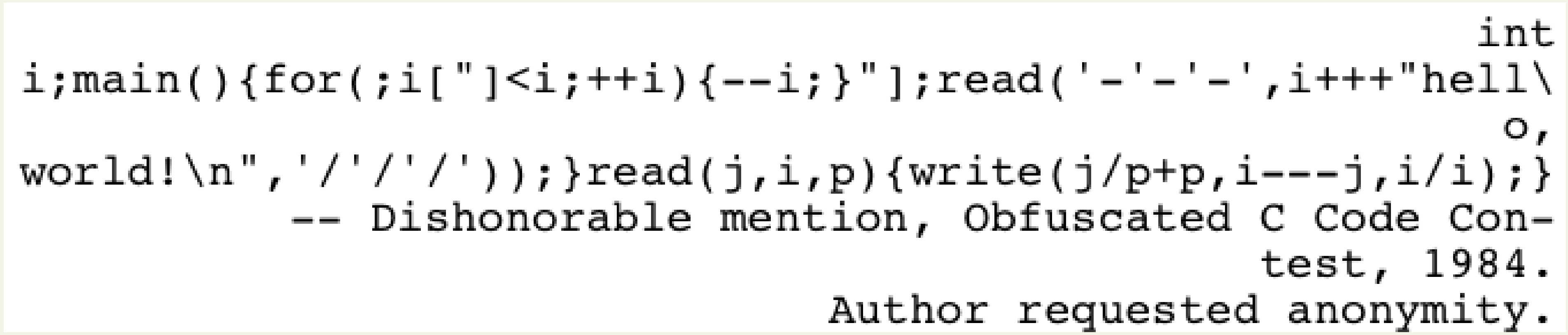
Suggested Programming Style for Lab Project

You are suggested to use the Programming style presented in this lecture

- It's a simplified version of GNU Coding Standards, with elements from the other references
- You may choose some variants, if with good reason

ALL partners of the same project team must use the same style with the same variants

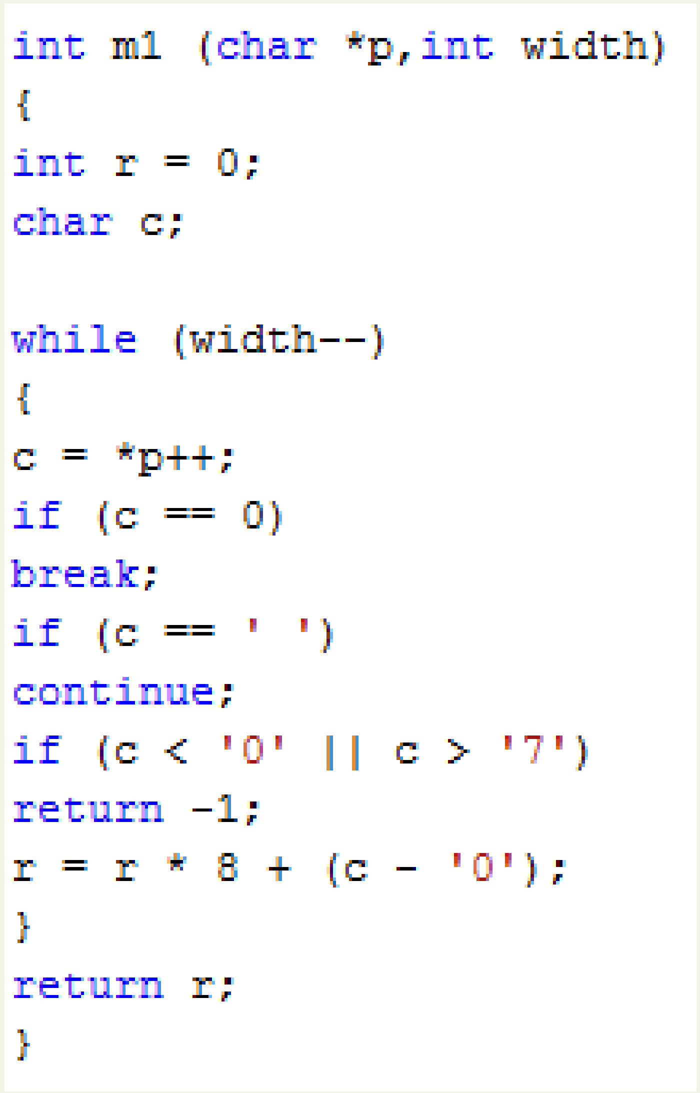
Why do we need it?



```
                                                         int
i;main(){for(;i["]<i;++i){--i;}"];read('-'-'-',i+++"hell\
                                                          o,
world!\n",'/'/'/'));}read(j,i,p){write(j/p+p,i---j,i/i);}
           -- Dishonorable mention, Obfuscated C Code Con-
                                                 test, 1984.
                               Author requested anonymity.
```

From “Recommended C Style and Coding Standards”

Why do we need it?

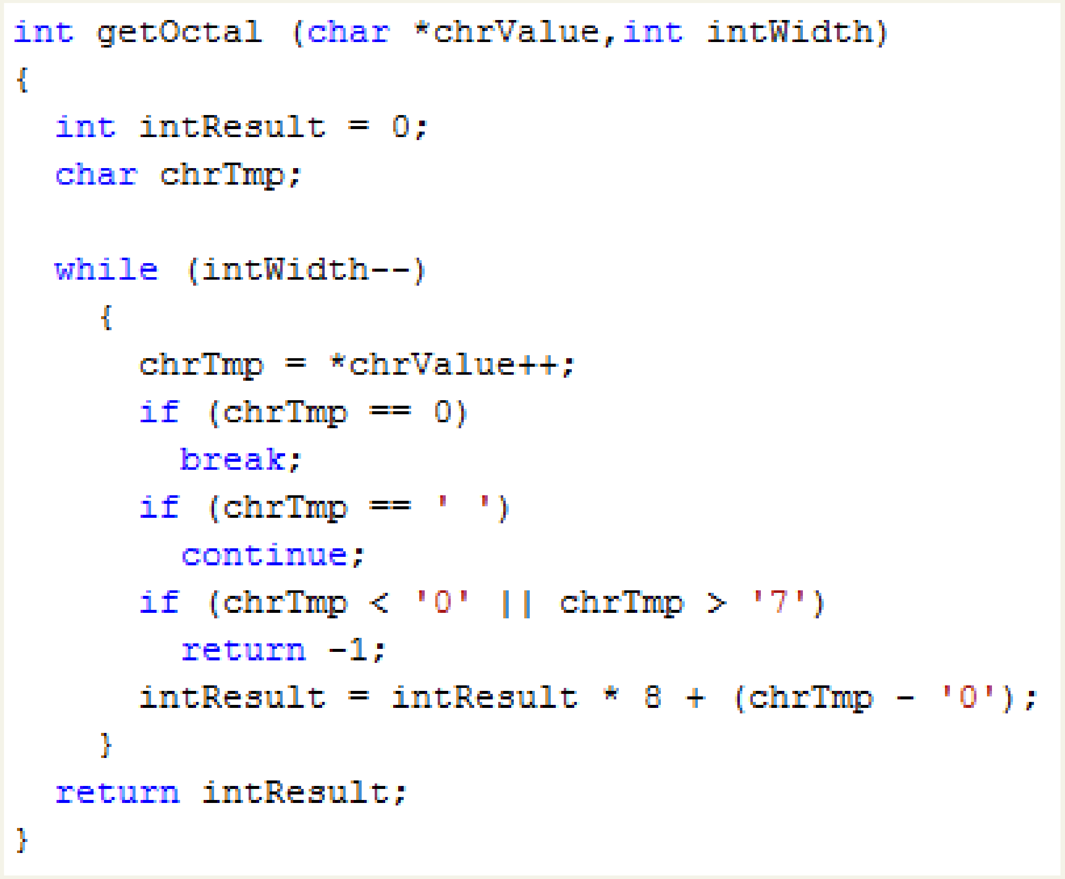


```
int m1 (char *p,int width)
{
int r = 0;
char c;

while (width--)
{
c = *p++;
if (c == 0)
break;
if (c == ' ')
continue;
if (c < '0' || c > '7')
return -1;
r = r * 8 + (c - '0');
}
return r;
}
```

Credit: Jafar M. Al-Kofahi

Why do we need it?



```
int getOctal (char *chrValue,int intWidth)
{
  int intResult = 0;
  char chrTmp;

  while (intWidth--)
    {
      chrTmp = *chrValue++;
      if (chrTmp == 0)
        break;
      if (chrTmp == ' ')
        continue;
      if (chrTmp < '0' || chrTmp > '7')
        return -1;
      intResult = intResult * 8 + (chrTmp - '0');
    }
  return intResult;
}
```

Credit: Jafar M. Al-Kofahi

Why do we need it?

We need a good coding style for many reasons

- Understand the code written by ourselves after some time
- Let others understand the code
- Reduce the number of bugs and the debugging time
- Overall, reduce the time spent on 288 Lab Project

C Programming Style

From GNU Coding Standards, Ch. 5, “Making the Best Use of C”

- Formatting: Format your source code
- Comments: Commenting your work
- Syntactic Convention: Clean use of C Constructs
- Names: Naming variables, functions, and files

Program File Layout

Suggested layout for .c files

1. A prologue that tells what is in the file
2. Any header file includes
3. Any defines and typedefs
4. Global data declarations
5. Functions, in some meaningful order

More details: Recommended C Style and Coding Standards, Sec 2.2 Program files

Program File Layout: Example

Prologue

```
/*
 * ping.c: Ping))) sensor related functions
 */
```

Includes

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include “servo.h”
```

Defines and defs

```
// Number of clock cycles for 1-meter distance (single-trip) under prescalar 256
#define   TICKS_PER_METER                    735
```

Global variables

```
volatile unsigned falling_time;              // captured time of falling edge
volatile unsigned rising_time;               // captured time of rising edge
```

Functions

```
unsigned ping_read()
{
   …
```

Header File Layouts

Use the same layout for .c program files, for declarations visible to outside

Use C Macro def to avoid nested includes

```
#ifndef EXAMPLE_H
#define EXAMPLE_H
... /* body of example.h file */
#endif /* EXAMPLE_H */
```

Use extern for global variable visible to outside

```
extern int sound_speed;
```

Format Function

GNU Function layout

- Brace starts at column 1 of a new line
- Function name starts at column 1 of a new line

```
static char *
concat (char *s1, char *s2)
{
    …
}
```

Format Expression

Break an long expression: Split it **before** an operator and align the two parts properly

```
if (foo_this_is_long && bar > win (x, y, z)
    && remaining_condition)
```

Extra parenthesis: Add extra parentheses if they can make expressions clearer

```
max = (x > y) ? x : y;
```

Indent Style: GNU

```
int
sample_func()
{
  while (x == y)
    {
      something ();

      if (some_error)
        do_correct ();
      else
        cont_as_usual ();
    }

  finalthing ();
}
```

GNU indent style

- The opening brace occupies a line
- The opening brace is indented by 2 spaces
- The next statement is indented by another 2 spaces

Indent Style: K&R

```
int sample_func()
{
    while (x == y) {
        something();

        if (some_error)
            do_correct();
        else
            cont_as_usual();
    }

    finalthing();
}
```

K&R indent style

- The opening brace of a control body does NOT take a line
- The next statement is indented by 4 spaces

The K&R Book: The C Programming Language, Brian W. Kernighan and Dennis M. Ritchie

Indent Style: Allman

```
int sample_func()
{
    while (x == y)
    {
        something();

        if (some_error)
            do_correct();
        else
            cont_as_usual();
    }

    finalthing();
}
```

Allman indent style (ANSI style)

- The opening brace of a control body takes a line
- The opening brace is indented by 0 space
- The next statement is indented by 4 spaces

Indent Style: Simple Control Statements

GNU:

```
if (x == y)
  do_something ();
else
  do_others ();
```

K&R and Allman:

```
if (x == y)
    do_something();
else
    do_others();
```

If the control body is a single statement:

- GNU: Indented by 2 spaces
- K&R and Allman: Indented by 4 spaces

GNU function call: Note the extra space between the function name and “(“

Indent Style: Lab Project

Which style to use? Your choice!

- Each style has its own rational and history

For the Lab Project

- GNU is more generous in using line space, more popular today because of GNU projects
- Allman is the most compatible, among the three, with the AVR's studio's default indentation
- K&R is the most compact, and more AVR-compatible than GNU

Everyone in the same team must use the same style!

Format Switch Statement

```
switch (expr)
  {
  case ABC:
  case DEF:
    statement;
    break;
  case UVW:
    statement;
  case XYZ:
    statement;
    break;
  }
```

GNU Style:

- Cases are aligned with the opening brace (indented by 2 spaces)
- The statements are indented by 2 spaces from case, 4 spaces from switch

Format Switch Statement

```
switch (expr) {
case ABC:
case DEF:
    statement;
    break;
case UVW:
    statement;
case XYZ:
    statement;
    break;
}
```

K&R Style

- Cases are aligned with the switch
- Statements are indented by 4 spaces from case and switch

Format Switch Statement

```
switch (expr)
{
case ABC:
case DEF:
    statement;
    break;
case UVW:
    statement;
case XYZ:
    statement;
    break;
}
```

Allman Style*

- Cases are aligned with the switch and the open brace
- Statements are indented by 4 spaces from case and switch

* This may not be the original Allman style

Format Statement

Automatic indent tool: indent

- Available on Linux, Mac or other UNIX-type systems

Format with the GNU style

```
indent -gnu sample.c
```

Format with the K&R style

```
indent -kr sample.c
```

Format with the original Berkeley style (also popular)

```
indent -orig sample.c
```

Commenting Your Work

GNU guidelines and our suggestion:

- Each program should start with a comment saying briefly what it is for
- Each function should have a starting comment saying what the function does
- Explain arguments properly, particularly if there is anything unusual
  - E.g. A string that is not necessarily zero-terminated
- Explain the return value
- Be generous in commenting, try to put a comment for every block of statements or statement with non-straightforward meaning

Commenting Your Work

More from “Recommended C Style and Coding Standards”

- Write a block of comment prologue to each function
- Make function return value have its own line, with probably a comment explain the return value (same as GNU)
- Try to align comments
- Use a blank line between local variable declarations and the function’s statements

Commenting: Example

```
/* Move serve to a angular position given by degree. */
void
move_servo(unsigned degree)
{
    unsigned pulse_width;          // pulse width in Timer/Counter cycles

    // Pulse width is (1+(degree/180))*t cycles, t is number of clock cycles
     per millisecond
    pulse_width = 1*MS_TICKS + (degree*MS_TICKS/180);

    OCR3B = pulse_width-1;                 // set pulse width
     wait_ms(500);                         // wait for half second for servo to
     settle
}
```

Commenting: Example

```
/* Start Ping))) sensor, read the pulse width, and return distance
   in millimeter */
unsigned distance //return distance, 0 if out of range (>1000mm)
ping_read()
{
    send_pulse();                   // send the starting pulse to PING
    state = LOW;                    // now in the LOW state

    // Enable Timer1 and interrupt, with noise cancellation
   (ICNC=1),
    // detecting rising edge (ICES=1), and prescalar 1024
   (CS=101)
    TCCR1B = _BV(ICNC) | _BV(ICES) | _BV(CS2) | _BV(CS0);
```

Commenting: Example

```
 // Wait until IC is done
 while (state != DONE)
      {}

 // Disable  Timer/Counter 1: CS=000
 TCCR1B &= ~(_BV(CS2) | _BV(CS1)| _BV(CS0));

// Convert time difference in cycles to distance in millimeter
unsigned dist = (falling_time - rising_time) / (2 *
cycles_per_mm);

// Out of range?
if (dist > 1000)
    dist = 0;

return dist;
}
```

Nested Control Statement

Always use braces to separate nested control statements

```
if (foo)
  {
    if (bar)
      win ();
    else
      lose ();
  }
```

The following style is bad

```
if (foo)
  if (bar)
    win ();
  else
    lose ();
```

Naming Conventions

GNU coding standards:

Use underscore to separate multiple words

```
falling_time
rising_time
init_servo
move_servo
```

Try to use short local variable names

Naming Conventions

More from "Recommended C Style and Coding Standards"

- Avoid local declarations that override declarations at higher level, e.g. local vs. global, same local names in nested blocks
- Avoid using names started with underscore (to avoid conflicts with system/library variables)
- #define constants should be in all CAPS
- Function, typedef, and variable names, as well as struct, union, and enum tag names should be in lower case
- Avoid names close to each other, e.g. foo and Foo, foobar and foo_bar, bl and b1 and bI (with upper case I)

White Space

Use white spaces generously

```
if ((a + b) == (c - d))
```

Split long for-loop and align the lines

```
for (curr = *listp, trail = listp;
     curr != NULL;
     trail = &(curr->next), curr = curr->next)
{
     ...
```

Program File Organization

Use multiple program files, one .c file and one .h file for each program module

Examples:

- lcd.c, lcd.h
- util.c, util.c
- ir_sensor.c, ir_sensor.h
- ping.c, ping.h
- robot.c, robot.h
- servo.c, servo.h
- main.c