EE 319K Homework Manual

Univ of Texas at Austin

Instructors: Gerstlauer, Telang, Valvano, Yerraballi

Do not print the entire document; we will be making changes.

Spring 2012 (4/23/12 version)

Homework Schedule

Due Date	Task
01/30/12	**Homework 1:** Hand Assemble
02/6/12	**Homework 2**: Assembly Concepts
02/13/12	**Homework 3**: Signed/Unsigned numbers Arithmetic/Logic operations (Patt, Chapters 11 and 12)
02/27/12	**Homework 4**: If-then-else, Loops, and Functions (Patt, Chapter 13 and 14)
03/7/12	**Homework 5**: Functions and Arrays (Patt, Chapters 14, 16.3)
03/19/12	**Homework 6**: Practice Exam 2s
03/26/12	**Homework 7**: Three programs in C using Arrays and parameter-passing
04/9/12	**Homework 8**: Metrowerks Introduction
04/23/12 (Monday)	**Homework 9**: Interpolation
04/30/12	**Homework 10**: Practice Final Exam

Homework 1: Introduction to Embedded Systems

EE 319K

The University of Texas at Austin

Ramesh Yerraballi

Due: Monday 1/30/12 11:00pm on Blackboard

Instructions:
You may discuss the problem with your fellow classmates but the write up must be your own. Please use the TAs and the Instructor for help before you seek out a friend or classmate.

Manually assemble the given program showing the address and machine code in hex for each instruction. Also, show the corresponding Symbol Table.

```
DDRH equ  $0262    ; Port H Data Direction Register
DDRT equu $0242    ; Port T Data Direction Register
PTH  equ  $0260    ; Port H I/O Register
PTT  equ  $0240    ; Port T I/O Register

     org  $4000    ; Object code goes in EEPROM
Main ldaa #$FF
     staa DDRT     ; Port T is output
     ldaa #$00
     staa DDRH     ; Port H is input
     ldab #129
     ldab #-127    ; Are these two instructions the same?
loop ldaa PTH      ; Read inputs
     staa PTT      ; Set output
     bra  loop         ; Repeat

     org $FFFE
     fdb Main      ; Starting address after a RESET
```

Also give the printout of the listing (`TheList` window) when you type and assemble this program in TExaS.

Do not cheat yourself, FIRST do the assembly by hand and then check it against the listing from TExaS. If there is a mismatch then review your hand assembled code to see where you may have gone wrong.

Submit: Upload a word or pdf file with one page of your hand assembled code and the second page that you cut-and-paste from TExaS listing.

Homework 2: EE 319K

Ramesh Yerraballi

Due: Monday 2/6/12 at 11:00pm on Blackboard

Instructions: Submit your answers electronically online on Blackboard. A single page (word or pdf document) with your answers will suffice.

This print-out should have 21 questions. Multiple-choice questions may continue on the next column or page – find all choices before answering.

JWV 2 1 Estimate Number
001 10.0 points

Using the estimation that 2^{10} is approximately equal to 10^3, without using a calculator, estimate the approximate value of 2^{28}. Do not give the exact value, but rather give the answer with one significant decimal digit. For example, 2^{18} is approximately equal to 300,000.

JWV 2 3 Convert8bitNumber3
002 (part 1 of 2) 10.0 points

Assuming an 8-bit unsigned integer format, what is the decimal value of hexadecimal `0xD5`?

003 (part 2 of 2) 10.0 points

Given the decimal number 162, what is its value in unsigned hexadecimal format?

1. 0x98

2. 0x8E

3. None of these

4. 0x9A

5. 0xB0

6. 0xA2

7. 0x93

8. 0xA8

JWV 2 3 Convert16bitNumber3
004 (part 1 of 2) 10.0 points

Assuming a 16-bit unsigned integer format, what is the decimal value of hexadecimal `0xB7B6`?

005 (part 2 of 2) 10.0 points

Given the decimal number 23730, what is its value in unsigned hexadecimal format?

1. 0x5BE9

2. 0x5C92

3. 0x5C5F

4. 0x5CF6

5. None of these

6. 0x5CB3

7. 0x5C49

8. 0x5C14

JWV 2 3 Convert8bitSNumber1
006 10.0 points

Assuming an 8-bit signed integer format, what is the decimal value of hexadecimal `$A2`?

JWV 2 4 Convert16bitSNumber1
007 10.0 points

Assuming a 16-bit signed integer format, what is the decimal value of hexadecimal **$8CF9**?

JWV 2 4 Number of bits1
008 10.0 points

How many binary bits are required to represent the number **70000**? For example, it takes 11 bits to represent the number 2000.

JWV 2 4 Number of bytes1
009 10.0 points

How many 8-bit bytes of storage are required to represent the number $\mathbf{9 \times 10^7}$? For example, it takes 2 bytes to represent the number 2000.

JWV 2 1 NumberFormat1
010 10.0 points

A software variable can take on whole number values from -200 to $+200$.

Which number format should be used for this variable? If more than one format is possible, choose the one with the best spaceefficiency.

1. 8-bit signed integer

2. 32-bit floating point

3. None of these choices is correct.

4. 16-bit signed fixed-point number with a resolution of 0.001

5. 16-bit signed fixed-point number with a resolution of 0.01

6. 16-bit signed integer

7. 32-bit signed integer

JWV 2 1 NumberFormat2

011 10.0 points

A software variable can take on whole number values from −100000 to +100000.

Which number format should be used for this variable? If more than one format could be used to solve the problem, choose the one with the most space-efficient format.

1. 16-bit signed fixed-point number with resolution of 0.01

2. 32-bit signed fixed-point number with resolution of 0.001

3. 32-bit floating point

4. 8-bit signed integer

5. 32-bit signed integer

6. None of these choices is correct.

7. 16-bit signed integer

JWV 2 8 Addition1

012 (part 1 of 5) 10.0 points

Consider the result of executing the following two 6811/6812 assembly instructions.

```
ldaa    #42
adda   #249
```

What is the value in Register A after these two instructions are executed? Give your answer in **unsigned decimal**.

013 (part 2 of 5) 10.0 points

What will be the value of the carry (C) bit? Give your answer as 0 or 1.

014 (part 3 of 5) 10.0 points

What will be the value of the overflow (V) bit? Give your answer as 0 or 1.

015 (part 4 of 5) 10.0 points

What will be the value of the zero (Z) bit? Give your answer as 0 or 1.

016 (part 5 of 5) 10.0 points

What will be the value of the negative (N) bit? Give your answer as 0 or 1.

JWV 2 8 Addition2

017 (part 1 of 5) 10.0 points

Consider the result of executing the following two 6811/6812 assembly instructions.

```
ldaa    #39
adda   #−9
```

What is the value in Register A after these two instructions are executed? Give your answer in **signed decimal**.

018 (part 2 of 5) 10.0 points

What will be the value of the carry (C) bit? Give your answer as 0 or 1.

019 (part 3 of 5) 10.0 points

What will be the value of the overflow (V) bit? Give your answer as 0 or 1.

020 (part 4 of 5) 10.0 points

What will be the value of the zero (Z) bit? Give your answer as 0 or 1.

021 (part 5 of 5) 10.0 points

What will be the value of the negative (N) bit? Give your answer as 0 or 1.

~~JWV 2 10 drop out~~

~~**022 10.0 points**~~

Homework 3

Due: Monday 2/13/12 11:00pm on Blackboard

Go to http://codepad.org and try out the following simple C Programs. When you visit the site you may choose to create an account that will help you track your work. It is not required to create an account to actually use the site though.

As I mentioned in the syllabus you should read the later parts of Dr. Patt's EE306 textbook for help with the basics of C. If you do not have the text or you are looking for more information, we have a C Primer for you that is specifically written for Embedded Systems (using Metrowerks) at the following URL:

http://users.ece.utexas.edu/~ryerraballi/CPrimer/

The concepts covered there apply to C programming in general but with a few exceptions like, the lack of input/output library support for functions like printf, scanf and no support for floating point types.

In this set we will look at declarations, logical operations, flow-control using conditionals (if-then-else statements). If you are referring to Yale Patt's book, you may want to read chapters 11, 12 and 13. If you are referring to the C Primer, you should read chapters 4, 5, and 6.

Exercise 0: The obligatory greeting program.

```
void main(void)
{
   printf("Hello 9S12");
}
```

Exercise 1: Numbers - Signed vs. Unsigned, 8-bit vs. 16-bit

In this exercise we will see how to define a 8-bit signed or unsigned number. Note that we use `char` as the type, to declare 8-bit numbers in C. To define a 16-bit number we use `short` as the type.

```
void main(void)
{
   unsigned char u8bit; // Unsigned 8-bit number
   signed char s8bit;   // Signed 8-bit number

   unsigned short u16bit;

   u8bit = 166;
   printf("u8bit as an unsigned number = %u\n",u8bit);
   printf("u8bit in decimal = %d\n",u8bit);
   printf("u8bit in hex = %x\n",u8bit);

   s8bit = u8bit;
   printf("s8bit as a signed number is = %d\n",s8bit);

   u16bit = 0xABCD; // Hex numbers have the 0x prefix
   printf("u16bit is = %d\n",u16bit);

}
```

`printf` is a library function you call in C programs to print something out to the screen. You pass it the string you want to output in quotes with placeholders for variables inserted where needed. The placeholders must have the format `%d` (for decimal), `%u` (for unsigned decimal), `%x` (for hexadecimal), `%c` (for character), `%s` (for string) etc. Other special characters like `\n` (for newline), `\t` (for tab) can also be embedded in the string.
Try different values for `u8bit`, `s8bit` and `u16bit` for practice.

(Note that in an embedded system we may not have a screen to output to so the `printf` library call may not exist)

Exercise 2: Logical Operations - AND, OR and XOR, left shift (<<), right shift(>>).
In this exercise we will see how to perform simple logical operations.

```
// Given switch1 is in bit position 1 of sw1
// Given switch2 is in bit position 0 of sw2
// Given the light is in bit position 7 of lt
// Logic: light ON only if both switches are ON
void main(void)
{
   unsigned char sw1, sw2, lt; //declare two switches and 1 light
// 1) Input phase of system
   sw1 = 0x02;   // change these values to try various
   sw2 = 0x01;   //   setting of the switches
// 2) Calculation phase
  lt = (sw1>>1 & sw2)<<7;  // l-shift to align the bits, r-shift for result

// 3) Output phase
   if (lt) {
       printf("Light On");
   }
   else {
       printf("Light Off");
   }
}
```

Try the code with different options for turning the light on:

Light ON if either of the two switches are ON. Note, the OR operator in C is the pipe symbol: "|"
Light ON if either of the two switches are ON but not both. Note, the XOR operator in C is the caret operator: "^"

Homework 3: Write code that counts the number of 1s in the odd positions of a given 8-bit number and checks it this matches the number of 1s in the even positions and turns a light on, otherwise it is off. Position 0 is considered even.
Here are a couple of examples:
Ex1: 8-bit number is 0xAA.
The number of 1s in the odd positions are 4,
the number in even positions is 0, so light *off*
Ex2: 8-bit number is 0x5A.
The number of 1s in the odd positions are 2,
the number in even positions is 2, so light *on*

Here is a skeleton to get you started:

```
// Number is given in num
// Assume the light is in bit position 4 of lt
#define test1 0xAA     // off
#define test2 0x5A     // on
#define test3 0x84     // on
#define test4 0xF0     // on
void main(void)
{
   unsigned char num, lt; //
// 1) Input phase of system
   num = test1;   // change test case to try your logic works
   // Write your code here to implement the logic described
// 2) Calculation phase

// 3) Output phase


}
```

Submit a printout of the Codepad screen showing both the code listing and the generated output. You will turn this in electronically on Blackboard. A screenshot of your codepad screen will suffice.

Homework 4

Due: Monday 2/27/12 11:00pm on Blackboard

In this set we will look at arithmetic and loops If you are referring to Yale Patt's book, you may want to read later part chapter 13. If you are referring to the C Primer, you should read later part of chapter 6.

Exercise 1: A simple count down using a while loop and the same concept implemented using a for loop

```
void main(void) {
   unsigned char count;
   count = 10; // Initialize the loop control variable, (count)
   while (count > 0) {
     printf("%d ",count);
     count = count - 1;  //update count and repeat
   }
   printf("Lift Off\n"); // When count becomes zero we get out of the loop

   for (count=10; count > 0; count--) {
       //count-- is short for count = count-1
    printf("%d ",count); printf("Lift Off\n");
   }
}
```

Write C code to do the following:

- Count down from 100 to 0 but print every other number out: 100, 98,96, ... 4, 2, 1 ---- Lift Off
- Print all numbers between 1 and 50 that are divisible by 3 or 5 (you will have to use a conditional statement in the body of the for loop)

Nested Loops: The body of a for or while loop may itself contain a for or while loop.

Exercise 2: Say we want to print the multiplication table for numbers between 2 and 9:

```
void main(void){
  unsigned char number, count, result;

  for (number = 2; number <= 9; number++) {
    printf("\nMultiplication Table for %d\n", number);
    for (count = 1; count <= 10; count++) {
      result = number*count;
      printf("%d X %d = %d\n", number, count, result);
   }
  }
}
```

Write C code to do the following:

• Print the factorials of numbers 1 to 15. The factorial of a number N is the N*(N-1)*(N-2) ... 1.

• You could have loops nested any number of times.

Print all Pythagorean triples less than 15. Refer to Figure 14.11 from Yale Patt's book

<u>Homework 4</u> : The solution to this exercise is what you will be submitting for this assignment. Write a subroutine which will be called by the main program to do the following with the input passed to it.
The subroutine iterates over all integers (inclusive) from 1 to the number passed to it. For example, if the input number is 10, the subroutine should iterate over 1 through 10. At each integer value in this range, your code may possibly (based upon the following rules) output a single string terminating
with a newline.
- For integers that are evenly divisible by three, output the exact string Hoppity, followed by a newline.
- For integers that are evenly divisible by five, output the exact string Hophop, followed by a newline.
- For integers that are evenly divisible by both three and five, do not do any of the above,
but instead output the exact string Hop, followed by a newline.
Here is code you are to complete:

```
void hoppityhop(unsigned char);
int main(){
   hoppityhop(10);
   hoppityhop(15);
   hoppityhop(55);
}
//Complete this subroutine according to the description given above.
// Your output will depend on the input and will be a sequence of
// lines that have either
// Hoppity, Hophop or Hop on them
void hoppityhop(unsigned char input) {




}
```

Here is the expected screen output for the call hoppityhop(15):

```
Input = 15
Hoppity
Hophop
Hoppity
Hoppity
Hophop
Hoppity
Hop
```

You will need to show the output for all three calls to the subroutine from main.

Submit a printout of the Codepad screen showing both the code listing and the generated output. You will turn this in electronically on Blackboard.
A screenshot of your codepad screen will suffice.

Homework 5

Due: Monday 3/5/12 11:00pm on Blackboard

You are required to print the results (screenshot) of the Homework 5 upload it as you did for homework 3. Go to http://codepad.org and try out the following simple C Programs and their suggested modifications.

In this set we will look at basics of arrays and revisit the for loop as a natural fit for traversing an array. We will also look at Subroutines and parameter passing. If you are referring to Yale Patt's book, you may want to read chapters 13, 14 and 16. If you are referring to the C Primer, you should read chapters 6, 7, 8 and 10.

Arrays

To declare an array of a particular type of size N we use the declaration:

```
     type arrayname[N];
// type indicates what each value in the array can be (for
example unsigned char).
// arrayname is the variable name by which the array will be
referred to
```

Examples:

```
    unsigned char anums[5];     // array of 5 8-bit unsigned
numbers
    signed char bnums[8];       // array of 8 8-bit signed
numbers
    unsigned short scores[25];  // array of 25 16-bit unsigned
numbers
```

To access individual elements of the array you use the square brackets with the index of the element you wish to access. Note that indexes start from 0. So, the first element's index is 0 and the last element's index is (Size-1).

<u>Exercise 1</u>: Lets say Sum(n) refers to the sum of the numbers n down to 1. For example (Sum(5) is 5+4+3+2+1 which equals 15). We wish to compute theses sums for the first 25 numbers and store them in an array Sum. We have another array FSum that computes these same sums by using a formula instead. We will check if the formula is correct by comparing elements of Sum against elements of FSum.

```
#define true 1
#define false 0
#define N 25     // The Size of the Array
void main(void){
  unsigned short num, partialsum, count;
  unsigned short Sum[N], FSum[N]; //Declared arrays to store 16-bit values
  unsigned char correct;

  for (num = 0; num < N; num++){
    partialsum = 0; // partialsum will hold the running sum as we compute it
    for (count = num+1; count >= 1; count--){
      partialsum += count;
    }
    Sum[num] = partialsum; //Note Sum[0] holds Sum(1); Sum[1] holds Sum(2) ar
  }

  for (num = 1; num <= N; num++){
    FSum[num-1] = (num * (num + 1))/2;
        // Note FSum[0] holds Sum(1); FSum[1] holds Sum(2) and so on
        // based on the formula Sum(n) = (n * (n+1))/2
  }

  // Check if the formula and computation agree
  correct = true; // Assume that they match and change it to false if
                  //  there is a mismatch
  for (num = 0; num < N; num++){
    if (Sum[num] != FSum[num]) {
      correct = false;
    }
  }
  if (correct) {
    printf("Formula Works");
  } else {
    printf("Formula Wrong");
  }
}
```

Write C code to do the following:

Declare 3 arrays called `bases`, `opposites` and `hypotenuses` all of the same size N. Populate theses arrays by computing the N Pythagorean triples that you wrote the code for in the previous exercise in homework 3.

Write a program that searches an array of N 16-bit numbers (call it `haystack`) to see if a particular value (call it `needle`) is present in it.   You can declare an array and initialize it in one step like so: `unsigned char foos[5] = {12, 4 , 13, 2, 15}; // foos[0] has 12, ... foos[4] has 15`

Subroutines (aka Functions)

Subroutines work in C very much like in assembly except they are called functions in C.

The key elements to note about functions are:

• Functions must be declared using function prototypes. This is used by the compiler when checking to see if you are calling a function correctly. For example the following function prototype says foofunc is a function that expects one unsigned short as input parameter and returns a signed char are output: `signed char foofunc(unsigned short); // Note only types need to the specified for checking`

- Calls to functions. A function call (like bsr or jsr in 9S12) must follow the calling convention specified by the prototype. So calls to foofunc for example have to be as follows: `result = foofunc(input); // input must be declared as an unsigned short and result as a signed char`
- Function implementations provide the code that will be executed when the function is called. The code captures the purpose of the function. Say foofunc simply returned the higher byte of the passed input as a signed number then the code would be as follows: `signed char foofunc(unsigned short input){ signed char result; result = input>>8; return result; }`

<u>Exercise 2</u>: Lets rewrite the code from Exercise 3 by modularizing it using subroutines:

```
#define true 1
#define false 0
#define N 25     // The Size of the Array

unsigned short Sum[N], FSum[N]; //Declared arrays to store 16-bit values
                                // These are now Global so all sub-routines i
                                // main can manipulate them

//Declare function prototypes here so compiler can check if we are
// calling them correctly
unsigned short hardway(unsigned char);
unsigned short formula(unsigned char);
void Check(void);

// The main program uses a modular decomposition of the problem into
// sub-problems that are implemented as functions that are called
int main(void){
  unsigned char num;

  for (num = 0; num < N; num++){
    Sum[num] = hardway(num); //Call the hardway function by passing it num
                               // put the returned sum in the corresponding l
                               // in the array Sum
  }

  for (num = 1; num <= N; num++){
    FSum[num-1] = formula(num);
  }
  Check();
}

unsigned short hardway(unsigned char number ){
  unsigned short sum = 0; // partialsum will hold the
                          //  running sum as we compute it
  unsigned char num, count;

  for (count = number+1; count >= 1; count--){
    sum += count;
  }
  return sum;   // sum holds the result so return it
}

unsigned short formula(unsigned char number ){

  return (number * (number+1))/2;   // formula

}

void Check(void){
  unsigned char correct, num;

  // Check if the formula and computation agree
  correct = true; // Assume that they match and change it to false if
                  //  there is a mismatch
  for (num = 0; num < N; num++){
    if (Sum[num] != FSum[num]) {
      correct = false;
    }
  }
  if (correct) {
    printf("Formula Works");
  } else {
    printf("Formula Wrong");
  }
}
```

Homework 5: Write a subroutine that counts the number of instances of letter in a string. All strings are 11 characters long (12 bytes including null.) Complete the implementation of the subroutine `Count`, and test it using the following main program. You may use pointer or index syntax to access data from the string. Take a printout from http://codepad.org showing your code and the output results of running your code. Turn in this printout (in word or pdf format) on Blackboard with your name at the top.

```
const struct countTestCase{
unsigned char Letter;        // Letter for which to search
unsigned char Buffer[12];    // String in which to search
unsigned short CorrectCount; // proper result of Count()
};
typedef const struct countTestCase countTestCaseType;

countTestCaseType countTests[7]={
{ 'o', "Hello World", 2},
{ 'b', "Bill Bard  ", 0},
{ 'V', "Jon Valvano", 1},
{ 'a', "Yerraballi ", 2},
{ 's', "Mississippi", 4},
{ '2', "21212121212", 6},
{ '1', "11111111111", 11}};

//This is the only subroutine you are expected to write.
// It is not necessary that you understand the rest of the code.
// What you need to know is what this subroutine is supposed to do.,
// that is, count the number of occurrences of letter in string.
unsigned short Count(unsigned char letter, unsigned char string[12]){
  return 1000; // replace this line with your code
}

int main(void){
  unsigned short i,result;
  unsigned short errors=0;
  for (i = 0; i < 7; i++){
    result = Count(countTests[i].Letter,countTests[i].Buffer);
    if (result != countTests[i].CorrectCount){
      errors++;
      printf("i=%d, result=%d\n",i,result);
    }
  }
  if (errors==0){
    printf("Program works");
  } else {
    printf("Does not work");
  }
  return 0;
}
```

Homework 6

Due: Monday 3/19/12 11:00pm on Blackboard

HW5 is a TExaS programming assignment and will serve as practice for Exam2. Please do ONE of the following three old Exam2s, upload the rtf file showing your name and score in comments at the top. Here are the exams to pick from:

http://users.ece.utexas.edu/~valvano/EE319K/HW6a.zip

http://users.ece.utexas.edu/~valvano/EE319K/HW6b.zip

http://users.ece.utexas.edu/~valvano/EE319K/HW6c.zip

Though I am only asking you to submit your solution to one of these three sample exams, I strongly urge you to solve all three of them to prepare for the exam.

Homework 7

Due: Monday 3/26/12 11:00pm on Blackboard

This homework is asking you to write the solutions for the three different Exam2-like problems. However, your solution is not in assembly, it must be in C. Attached you will find three different C programs that have a incomplete subroutine that you are expected to complete.

Read the comments on top the subroutine to see what you are expected to do.

```
//*************** HW 6 Part 1 ********************
// You have to write the code for ChkPalind
// You may call the routine PtrLast inside ChkPalind

#include <stdio.h>

unsigned char * PtrLast(unsigned char *);

const struct TestCase{
unsigned char *Buffer;    // String to check; Null terminated
signed char check; // +1 or -1 according as Buffer is or is not a
palindrome
};

typedef const struct TestCase TestCaseType;

TestCaseType Tests[5]={
{ "Madam\0",-1 },
{ "risetovotesir\0",1 },
{ "raceCar\0",-1 },
{ "NeveroddoreveN\0",1 },
{ "\0",1 }};

//This is the only subroutine you are expected to write.
// It is not necessary that you understand the rest of the code.
// What you need to know is what this subroutine is supposed to
do.,
// that is, check if the given string is a palindrome and return
a 1 if
// it is OR a -1 if it is not
// Input: Pointer to a null-terminated string of characters
// Output: 1 or -1 depending on whether the given string is a
palindrome
//         or not

unsigned char ChkPalind(unsigned char *string){
  return(0); //replace this line with your code
}

// This routine is given to you
// It takes a pointer to a string and returns the
// pointer to the last (non-null) character in the string

unsigned char * PtrLast(unsigned char *string){
  unsigned char *work;

  work = string;
  while(*work != '\0'){ work++;}
  if (work != string) {
    work--; // want the ptr to the last char so decrement
  }
  return(work);
}

int main(void){
  signed char i,result;
  unsigned char errors=0;

  for (i = 0; i < 5; i++){
    result = ChkPalind(Tests[i].Buffer);
    if (result != Tests[i].check){
      errors++;
      printf("Error Case: i=%d, String= %s, result=%d\n",i,
Tests[i].Buffer, result);
    }
  }
  if (errors==0){
    printf("Program works\n");
  } else {
    printf("Does not work\n");
  }
  return 0;
}

//*************** HW 6 Part 2 *******************
// You have to write the code for the function OddSum
#include <stdio.h>

unsigned short OddSum(unsigned char);

const struct TestCase{
unsigned char N;
unsigned short oddsum;
};

typedef const struct TestCase TestCaseType;

TestCaseType Tests[5]={
{ 12,36 },
{ 7,16 },
{ 0,0 },
{ 40,400 },
{ 1,1 }};

//This is the only subroutine you are expected to write.
// It is not necessary that you understand the rest of the code.
// What you need to know is what this subroutine is supposed to
do.,
// that is, return the sum of all odd numbers from 1 till N.
// N may or may not be included depending on whether it is odd or
even
// Input: The number N
// Output: sum of all odd numbers from 1 to N

unsigned short OddSum(unsigned char input){
  return(100); //replace this line with your code
}

int main(void){
  signed char i;
  unsigned char errors=0;
  unsigned short result;

  for (i = 0; i < 5; i++){
    result = OddSum(Tests[i].N);
    if (result != Tests[i].oddsum){
      errors++;
      printf("Error Case: i=%d, N= %d, result=%d\n",i, Tests[i].N,
result);
    }
  }
  if (errors==0){
    printf("Program works\n");
  } else {
    printf("Does not work\n");
  }
  return 0;
}

//************** HW 6 Part 3 *******************
// You have to write the code for the function IntvlCount
#include <stdio.h>

unsigned char IntvlCount(signed char *array, signed char llim,
signed char ulim);

const struct TestCase{
  signed char list[50];
  signed char intvl[2];
  unsigned char count;
};
```

```
typedef const struct TestCase TestCaseType;

TestCaseType Tests[5]={
  { {4,12,19,2,6},{3,9},1 },
  { {3,-1,0,4},{-1,1},2 },
  { {5,-2,3,15,62,-15},{4,10},0 },
  { {3,0,-1,1},{-1,1},3 },
  { {3,2,5,7},{8,10},0 }
};

//This is the only subroutine you are expected to write.
// It is not necessary that you understand the rest of the code.
// What you need to know is what this subroutine is supposed to
do.,
// that is,
// In the given array find how many elements fall in the interval
// covered by [llim,ulim] with boundaries included. Note that the
// input array has size as the first element.
// Input: Array of 8-bit signed numbers with the first element
being the size
//        llim is the lower limit of the interval to check
//        ulim is the upper limit of the interval to check
// Output: Return the number of elements in array that fall
within the interval
//        covered by [llim,ulim]

unsigned char IntvlCount(signed char *array, signed char llim,
signed char ulim){
  return(100); //replace this line with your code
}

int main(void){
  unsigned char i, errors=0, result;

  for (i = 0; i < 5; i++){
    result =
IntvlCount(Tests[i].list,Tests[i].intvl[0],Tests[i].intvl[1]);
    if (result != Tests[i].count){
      errors++;
      printf("Error Case: i=%d, result=%d\n",i, result);
    }
  }
  if (errors==0){
    printf("Program works\n");
  } else {
    printf("Does not work\n");
  }
  return 0;
}
```

Homework 8

Due: Monday 4/9/12 11:00pm on Blackboard

You will view, understand and implement the example shown in Lesson 5 of Jon's lessons on Metrowerks. First, you will need to download and install the Metrowerks Codewarrior IDE on your PC by following the instructions provided here: http://users.ece.utexas.edu/~valvano/S12C32.htm#Metrowerks

The Lessons on Metrowerks (there are 6 of these) are here: http://users.ece.utexas.edu/~valvano/Lessons/#Metrowerks

The lessons are incremental in that they start by showing you how to use Metrowerks to write C code, assembly code, mix C and assembly, pass parameters from C to assembly and vice versa and finally, how to run code that was developed in Metrowerks, in TExaS.

For this homework you will view lessons 1 through 4 and write the code that lesson 5 refers to. You will not submit your code though, instead you will submit a word document with the five screenshots that you take at points in the development process. Specifically, the five screenshots are:

1. A screenshot of the C code in Metrowerks with the comments at the top showing your name
2. A screenshot of the assembly code in Metrowerks with the comments at the top showing your name
3. A screenshot of the running of your code in "Full chip simulation" in the True-time simulator with the Data window showing the variable "MyCount" before a call to PortP_Toggle
4. A screenshot of the running of your code in "Full chip simulation" in the True-time simulator with the Register window showing the contents of register D once you enter the assembly routine PortP_Toggle.
5. A screenshot of the running of your code in "Full chip simulation" in the True-time simulator with the Register window showing the contents of register B once you enter C routine Toggle.

You will submit a single word file with the five screenshots in it.

Homework 9

Signal Interpolation

Due: Monday 4/23/12

The purpose of this assignment to give you some insight into how the technique of interpolation can be used to estimate values a signal takes between two known data points. Also, we will further exercise parameter-passing in C using arrays. The particular interpolation mechanism we will use is called *Cubic Interpolation*. The idea behind this technique is given in the following document online:
http://paulbourke.net/miscellaneous/interpolation/

The starter code with comments can be found at the following link:
http://users.ece.utexas.edu/~ryerraballi/ee319k/HW9.c

Take a printout from http://codepad.org showing your code and the output results of running your code. Turn in this printout (in word or pdf format) on Blackboard with your name at the top.

Homework 10

Due: In your class after Monday 4/30/12

For this homework, you are required to solve one final exam from a previous semester. You may choose from the following:

Valvano's previous exams:

1. Fall 2009:
   http://users.ece.utexas.edu/~valvano/EE319K/FinalF09a.pdf
2. Spring 2010:
   http://users.ece.utexas.edu/~valvano/EE319K/FinalSp10b.pdf
3. Fall 2010:
   http://users.ece.utexas.edu/~valvano/EE319K/FinalF10a.pdf
4. Spring 2011:
   http://users.ece.utexas.edu/~valvano/EE319K/FinalSp11a.pdf

Yerraballi's previous exams:

1. Spring 2010:
   http://users.ece.utexas.edu/~ryerraballi/ee319k/finals/S10.pdf
2. Fall 2010:
   http://users.ece.utexas.edu/~ryerraballi/ee319k/finals/F10.pdf
3. Spring 2011:
   http://users.ece.utexas.edu/~ryerraballi/ee319k/finals/S11.pdf
4. Fall 2011:
   http://users.ece.utexas.edu/~ryerraballi/ee319k/finals/F11.pdf

You are required to submit a hand-written solution to one of the above eight exams, in class. The due date is either Monday or Tuesday depending on which professor you are taking the class with and the particular lecture section you are registered in.