ME 333: Introduction to Mechatronics

Solutions to Assignment 1

January 19, 2012

Questions

1) What is 333_{10} in binary and 1011110111_2 in hexadecimal? What is the maximum value, in decimal, that a 12-bit number can hold?

   333_{10} in binary is 101001101_2, 1011110111_2 is $2F7_{16}$ (or 0x2F7) in hex, and the maximum value a 12-bit number can hold is 4,095 ($2^{12} - 1$).

2) What is the range of values for an `unsigned char`, `short`, and `double` data type? How many bytes of memory does each data type use? Assume the typical data sizes mentioned in this appendix.

type	min	max	memory usage (bytes)
unsigned char	0	255	1
short	-32,768	32,767	2
double	-1.797...e+308	1.797...e+308	8

3) In each snippet of code in (a)-(d), there is an arithmetic error in the final assignment of `ans`. What is the final value of `ans` in each case?

   (a)
   ```
   char c = 17;
   float ans = (1 / 2) * c;
   ```
   The final value of `ans` is zero. This result is due to a truncation error.

   (b)
   ```
   unsigned int ans = -4294967295;
   ```
   In this case, `ans = 1` should be the outcome, but the final value may depend on your compiler. It's OK if you typed this line into a C program and printed out the result. The moral of the story, don't use negative numbers with unsigned data types. The result can be unexpected.

   If you got `ans = 1`, the answer seems a little weird, but here is why it's normal from a computer's perspective. In two's complement (a representation for signed integers on nearly all computers), the bit representation of $-4,294,967,295_{10}$ is $100000000000000000000000000000001_2$, note that this number requires 33 bits. We know that an unsigned int is 32 bits, so we can ignore the MSB. This is what most computers do when there is an overflow with an unsigned integer; they typically just throw away the higher order bits that don't fit into the destination data type. Doing so, we get 1, which is the final value of `ans`. If your compiler assumed a sign and magnitude representation, then you would get a different result. If you are curious, see if you can work out what `ans` would be using sign and magnitude.

   (c)
   ```
   double d = pow(2, 16);
   short ans = (short) d;
   ```
   In this case, `ans = -32,768`, but it might be different on some computer platforms. Here again you should be aware about the dangers of using the incorrect data type for the range of values you are expecting to accept in your program.

   (d)
   ```
   double ans = ((double) -15 * 7) / (16 / 17) + 2.0;
   ```
   Because we have a division by zero, `ans = -INF`. (Depending on the compiler, you might get a different value, such as -1.#INF with the Microsoft Visual Studio compiler.)

4) Truncation isnt always bad. Say you wanted to store a list of percentages rounded down to the nearest percent, but you were tight on space and cleverly used an array of `char`s to store the values. For example, pretend you already had the following snippet of code:

```
char percent(int a, int b)
{
  // assume a <= b
  char c;
  c = ???;
  return c;
}
```

You can't simply write `c = a / b`. If $\frac{a}{b} = 0.77426$ or $\frac{a}{b} = 0.778$, then the correct return value is $c = 77$. Finish the function definition by writing a one line statement to replace `c = ???`.

There are many ways to answer this question. Some solutions require you to memorize the order in which an expression will evaluate in, but others use parentheses to control the order of evaluation. The following give equivalent results:

```
c = (100 * a) / b;
c = 100.0 * a / b;
c = 100 * (a / (b * 1.0));
c = 100 * ((float) a) / b;
c = 100 * (((float) a) / ((float) b));
```

and so on. In these examples, the compiler does an implicit cast to the `char` data type before the assignment. If you wanted to be explicit about it, you can replace any of the expressions with an explicit cast to `char`, for example, `c = (char) (100 * ((float) a) / b);`.

5) Consider the following lines of code:

```
int i, tmp, *ptr, arr[7] = {10, 20, 30, 40, 50, 60, 70};

ptr = &arr[6];
for(i = 0; i < 4; i++) {
  tmp = arr[i];
  arr[i] = *ptr;
  *ptr = tmp;
  ptr--;
}
```

(a) How many elements does the array `arr` have?
The array has seven elements.

(b) How would you access the middle element of `arr` and assign its value to the variable `tmp`? Do this two ways, once indexing into the array and the other with the dereferencing operator and some pointer arithmetic. Your answer should only be in terms of the variables `arr` and `tmp`.
Indexing: tmp = arr[3];
Pointer arithmetic: tmp = *(arr + 3);

(c) What are the contents of the array `arr` before and after the loop?
Before the loop `arr` = {10, 20, 30, 40, 50, 60, 70} and after the loop it's the reverse, `arr` = {70, 60, 50, 40, 30, 20, 10}.

6) The following questions pertain to a version of the Simple2.c program, reproduced below. For your responses, you only need to write down the changes you would make using valid C code. You should verify that your modifications actually compile and run correctly. Do not submit a C program for this question. We only want you to write down the changes you would make using legitimate C syntax.

```
#include <stdio.h>

#define MAX 100

void MyFcn(int max);

void MyFcn(int max)
{
  int i;
  double arr[MAX];

  if(max > MAX) {
      printf("The range requested is too large.  Max is %d.\n", MAX);
      return;
  }

  for(i = 0; i < max; i++) {
      arr[i] = 0.5 * i;
      printf("The value of i is %d and %d/2 is %f.\n", i, i, arr[i]);
  }

}

int main(void)
{
  MyFcn(5);
  return(0);
}
```

(a) `while` loops and `for` loops are essentially the same thing. How would you write an equivalent while loop that replicates the behavior of the for loop?

```
i = 0;
while(i < max) {
  arr[i] = 0.5 * i;
  printf("The value of i is %d and %d/2 is %f.\n", i, i, arr[i]);
  i++;
}
```

(b) How would you modify the main function, so that it reads in an integer value from the keyboard and then passes the result to `MyFcn`? (This replaces the statement `MyFcn(5);`.) If you need to use extra variables, make sure to declare them before you use them in your snippet of code.

```
int input;
printf("Please enter an integer for this demo: ");
scanf("%d", &input);
MyFcn(input);
```

(c) Change main so that if the input value from the keyboard is between `-MAX` and `MAX`, you call `MyFcn` with the absolute value of the input. If the input is outside this range, then you simply call `MyFcn` with the value `MAX`. How would you make these changes using conditional statements?

```
int input;
printf("Please enter an integer for this demo: ");
scanf("%d", &input);
```

```
if((input >= -MAX) && (input < 0)) {
  // input is valid, but negative.  Flip the sign.
  MyFcn(-input);
}
else if((input >= 0) && (input <= MAX)) {
  // pass input unchanged
  MyFcn(input);
}
else {
  // input is outside of the expected range
  MyFcn(MAX);
}
```

(d) In C, you will often find yourself writing nested loops (a loop inside a loop) to accomplish a task. Modify the `for` loop to use nested loops to set the i^{th} element in the array `arr` to half the sum of the first i integers starting from zero, i.e., `arr[i]` $= \frac{1}{2}\sum_{j=0}^{i-1} j$. (You can easily find a formula for this that doesnt require the inner loop, but you should use nested loops for this problem.) The same loops should print the value of each `arr[i]` to 2 decimal places using the %f formatting directive.

```
int j; // second counter variable
int tmp; // temporary variable to store the sum.
for(i = 0; i < max; i++) {
  // clear the sum from the previous run
  tmp = 0;
  // calc the sum
  for(j = 0; j <= i; j++) {
     tmp += j;
   }
   arr[i] = 0.5 * tmp;
   printf("The value of arr[%d] is %1.2f.\n", i, arr[i]);
 }
```

7) not assigned.

8) not assigned.

9)
```
/*******************************************************************************
* Jarvis Schultz
*
* January 5, 2012
*
* This code is an idea for a homework problem for ME333 in Winter of 2012.
* The problem was stolen from a "puzzler" I heard on NPR's CarTalk program.
*
* Copied directly from their website at:
*      http://www.cartalk.com/content/hall-lights?question
*
* RAY: This puzzler is from my "ceiling light" series. Imagine, if
* you will, that you have a long, long corridor that stretches out as
* far as the eye can see. In that corridor, attached to the ceiling
* are lights that are operated with a pull cord.
*
* There are gazillions of them, as far as the eye can see. Let's say
* there are 20,000 lights in a row.
```

```
 *
 * They're all off. Somebody comes along and pulls on each of the chains,
 * turning on each one of the lights. Another person comes right behind,
 * and pulls the chain on every second light.
 *
 * TOM: Thereby turning off lights 2, 4, 6, 8 and so on.
 *
 * RAY: Right. Now, a third person comes along and pulls the cord on
 * every third light. That is, lights number 3, 6, 9, 12, 15,
 * etcetera. Another person comes along and pulls the cord on lights
 * number 4, 8, 12, 16 and so on. Of course, each person is turning on
 * some lights and turning other lights off.
 *
 * If there are 20,000 lights, at some point someone is going to come
 * skipping along and pull every 20,000th chain.
 *
 * When that happens, some lights will be on, and some will be off. Can
 * you predict which lights will be on?
 *
 *******************************************************************************/


/************/
/* INCLUDES */
/************/
#include <stdio.h> /* We include this library so that we have access
    * to some input and output functions such as
    * 'printf' and 'scanf'. */

#include <stdlib.h>  /* We are going to include this library because
      * we want to use the 'exit' function later to
      * quit the program in the event of an error */



/***********************/
/* FUNCTION PROTOTYPES */
/***********************/
/* To solve this problem, we will use two functions one for actually
 * tracking the status of the lights as they are toggled, and another
 * for printing the results.  So let's write those prototypes here:*/
void toggle_lights(int* lights, int num);
void print_results(int* lights, int num);


/********************/
/* GLOBAL VARIABLES */
/********************/
/* Since we are allowing a variable number of lightbulbs, let's define
 * a constant (using a '#define') that sets the maximum number of
 * lightbulbs we are going to deal with.  This is important because we
 * don't want the user to be able to select an extremely large number
 * and overflow our system's memory. */
#define MAX_LIGHTBULBS (1000000) /* A million lightbulbs should be
  * fine on most modern computers */
```

```
/*****************/
/* MAIN FUNCTION */
/*****************/
void main(void)
{
  /* We begin our program by declaring all of the variables that
   * 'main' function will be using. */

  /* declare an int variable to represent the total number of
   * lightbulbs that we are dealing with:*/
  int num = 0;
  /* create a large int array (of size MAX_LIGHTBULBS) that will
   * contain the status of each lightbulb. */
  int lights[MAX_LIGHTBULBS];


  /* The next step in the program is to ask the user for how many
   * lightbulbs they are interested in analyzing. */

  /* provide some input telling the user we want them to enter a number */
  printf("Please enter the total number of lightbulbs that you want to \r\n"
 "analyze, and then press enter\r\n");
  /* wait for their input, and once we recieve it, store it in the
   * previously defined variable */
  scanf("%d", &num);
  /* now echo the input back to the user so that they know we recieved
   * it correctly. */
  printf("\r\nYou said %d lightbulbs.\n\r\r\n", num);


  /* Now that we know how many lightbulbs we are going to analyze,
   * let's perform a check with an 'if' statement to be sure that the
   * user did not input a number that is too large.  If they did,
   * let's tell them that and 'exit' the program. */
  if (num > MAX_LIGHTBULBS)
  {
    printf("ERROR: the input number is larger than the maximum value!\n\r");
    exit(1);
  }


  /* Since everything is ok with the input value let's call the
   * function that performs all of the light toggling */
  toggle_lights(lights, num);


  /* Now let's print out the results by calling our printing function. */
  print_results(lights, num);

  return;
}


/**********************/
/* CALLABLE FUNCTIONS */
/**********************/
```

```
/* This function takes in a pointer to a light array, and the total
 * number of lights we are concerned with.  It performes the toggling
 * duties that are outlined in the problem statement and returns
 * nothing. */
void toggle_lights(int* lights, int num)
{
  /* define an integer to set the current step size: */
  int dl = 1;
  /* define an integer to mark our current place in the light
   * array */
  int l = 0;

  /* We have a pointer to a light array, but we have no idea what the
   * current status of each lightbulb is.  To be sure that we know the
   * initial state of each lightbulb, let's use a 'for' loop to set
   * all of the initial values.  We will use a numeric value of 0 to
   * indicate the light is off, and a numeric value of 1 to indicate a
   * light is on. */
  for(l=0; l<num; l++)
    lights[l] = 0;

  /* Now we are ready to perform the toggling.  We will accomplish
   * this with a double 'for' loop. */

  /* the outer 'for' loop increments through the step size */
  for(dl=1; dl<=num; dl++)
  {
    /* the inner loop steps through all of the lightbulbs with a step
     * size of 'dl' and toggles each light's status. */
    for(l=dl-1; l<num; l+=dl)
    {
      /* for each light, let's toggle its value */
      lights[l]  = !lights[l];
    }
  }

  return;
}


/* This function prints out a list of the lights that are on, and the
 * total number of lights that are on. */
void print_results(int* lights, int num)
{
  int j = 0; /* integer marker for stepping through the light array */
  int count = 0; /* total count of lights that are on */

  printf("Printing results:\r\n");

  /* Following for loop will step through each light and determine
   * if it is on. */
  for(j=0; j<num; j++)
  {
    /* Is the current light on? */
    if(lights[j] == 1)
    {
      /* If so, let's print that out. Note that we are going to
       * add one to the index to account for the fact that the
       * zeroth entry in the array is the first lightbulb. */
      printf("Light number %i is on\r\n",
     j+1);
      /* Also, increment the total count. */
      count++;
    }
  }

  /* Print final total count. */
  printf("There are %i total lights on!\r\n", count);

  return;
}
```