NCSU ECE 209 Sections 602
Final Exam Fall 2009
3:00 PM – 6:00 PM
10 December, 2009

This is a closed book and closed notes exam. It is to be turned in by 6:00 pm. Calculators, PDA's, cell phones, and any other electronic or communication devices may not be used during this exam.

Please read and sign the following statement:

I have neither given not received unauthorized assistance on this test.

Name: _____

If you want partial credit for imperfect answers, explain the reason for your answer!

Problem 1 (10 points)
Complete the following C program so that it writes your *clearly spelled* name once.

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
  printf("My name is %s\n",

         "                                  ") ;
  return(EXIT_SUCCESS) ;
}
```

Problem 2 (5 points)

What does the `switch` statement, with the `int` variable `n`, on the right print when

1)`n` is 0?

2)`n` is 2?

3)`n` is 4?

```
switch(n) {
  case 1:
    printf("single\n") ;
    break ;
  case 2:
    printf("double\n") ;
  case 3:
    printf("triple\n") ;
    break ;
  case 4:
    printf("home run\n") ;
    break ;
  default:
    printf("huh?\n") ;
}
```

Problem 3 (15 points)

Below is a sequence of C statements defining some variables

```
int    vi  = 6 ;
char   uni = 'U' ;  /* ASCII code for 'U' is 85 */
double pin = 3.1 ;
```

The table below contains two columns. The leftmost column is a C expression. In the rightmost column write the value of the expression as a C literal. If the value is a `double`, be sure to use the C syntax for writing doubles. Some of these are tricky, but none require complex arithmetic.

`uni + 2`	
`(int)uni`	
`pin + vi`	
`++uni`	
`uni++`	
`0 * vi * vi`	
`0 * pin * pin`	
`1 < 14`	
`1 < 14 + 2 * 2`	
`1 < 1`	
`1 << 1`	
`3.0 * 0`	
`3/2`	
`3%2`	
`0 && (1 || ~(0 && 1))`	
`5 & 1`	
`5 && 1`	
`1 || 0`	
`1 || 1000`	
`5/10/2`	
`5*2/10`	
`5/2*10`	
`1.6 + 1.6`	
`(int)1.6 + 1.6`	
`(int)1.6 + (int)1.6`	
`(int)(1.6 + 1.6)`	

Problem 4 (10 points)
Each of the following four `for` or `while` loops prints numbers. For each loop (which are often proceeded by a few initializations) write in the four boxes below the loop the first four lines printed by loop. If the loop prints less than four lines, fill in a box for each line that is printed.

Assume that `i` and `j` have already been declared as `int` variables before each loop.

```
j = 13 ;
for (i=1; j>10; ++i) {
  printf(”%d\n”, i) ;
  j = j – 1 ;
}
```


```
for (i=0, j=1; i<10; i=i+j, j=j+2)
  printf(”%d\n”, i) ;
```


```
i=12 ;
while(i%2 == 0) {
  i =  i/2 ;
  printf(”%d\n”, i) ;
}
```


```
i = 0 ;
while ((i=i+5)< 10)
  printf(”%d\n”, i) ;
```


The remaining problems of the final exam involve writing sections of C programs. Your answers for these problems should be *neatly* written, *with appropriate indentation*, on separate sheets of lined paper, preferably on one side of the paper.

In your answer, you do not need to have any `#include` statements, just assume the appropriate ones are there. However, functions *must* start with appropriate C headers.

We'll start with a couple of CodeLab inspired problems.

Problem 5 (10 points)
First, define a structure whose tag (or `struct`) name is `Employee` and that contains three fields: a `double` named `Salary`, a character pointer named `Name`, and an array of seven integers named `Hours`.

Next, declare an array named `Staff` to contain 30 of these `Employee` structures.

Finally, make the following changes to the `Staff` array.

1) Assign the value 6 to the last element of the `Hours` field of the last element of `Staff`.
2) Assign 3.5 to the `Salary` field of the first element of `Staff`.
3) Assign the string "`George`" to the `Name` field of the `k`'th element of Staff. Assume that `k` is an `int` variable that has been given a value between 5 and 25.

Problem 6 (10 points)
Write a function named `negateOdd` that takes two arguments. The first is an array of integers. The second is the number of integers in the array. Your function should negates (changes the sign) of every odd integer in the array and returns the number of odd integers.

Problem 7 (5 points)
Write a short section of C code that tries to open a file called `ECE209.DAT` for reading. If the open fails, the program should print the message "`Unable to read ECE209.DAT`". If the open succeeds no message needs to be printed.

Problem 8 (5 points)
Suppose a `struct Name` is defined as follows.

```
struct Name {
     char LastName[25] ;
     char FirstName[15] ;
} ;
```

Write a function called `SameName` that receives as its two arguments pointers to structures declared to be `struct Name`'s. The function should return 1, if the two structures are the same when their corresponding fields are compared as strings; and should return 0, otherwise. Hint: Use `strncmp`.

The header for this function should be:

```
int SameName(struct Name *N1, struct Name *N2)
```

Problem 10 (20 points)
Write a program that reads a series of integers similar to that seen below.

```
    209   1776    13   -1   -13
        1066
       3   0    -15        209
```

When the program has finished reading the input (that is, has read to the end-of-file), it should print (1) the number of positive integers in the file, (2) the number of negative integers in the file, (3) the proportion of the integers that are positive, and (4), the average of the positive integers. If make things a bit simpler, assume that zero is a positive integer. For the above example, the output should look something like:

```
    Positive:         7
    Negative:         3
    Proportion positive:        0.70
    Average of positive:      468.00
```

Your output does not have to look exactly like this, but it should be neat.

In your program you may assume that all numbers are within the range of `int`'s for your C implementation. You may also assume that everything in the input is either a valid integer or white space separating valid integers.

Problem 11 (10 points)
Hopefully, you recall the definition of the `RLVariableNode` structure from Assignment 7.

```
struct RLVariableNode {
  struct RLVariableNode *next ;       /* Link to next in chain */
  struct RLAssignmentNode *variable ; /* Link to variable */
  double value ;                      /* Value of variable */
} ;
```

In the final question of this final exam, assume, as with Assignment 7, that `dummyNode` is a pointer to a `RLVariableNode` structure that serves as the "dummy" for a circular linked list of `RLVariableNode`'s. Don't worry about how `dummyNode` is set, just assume the following.

```
    struct RLVariableNode *dummyNode = MagicYouDontWorryAbout ;
```

We'll not use the `variable` field in these questions. That way there will be no concern about what a `struct RLAssignmentNode` is.

First, write a C loop to follow the following statement:

```
    struct RLVariableNode *presNode = dummyNode->Next ;
```

The C loop should set `presNode` to point to the first node of the list with a negative value. If no negative value is found before `dummyNode` is re-encountered, set `presNode` to `dummyNode`.

Now let's "remove" all those starting negative values by making `dummyNode` point to your new value for `presNode`.

Finally, "talk" a little about how you might remove all negative values from the list, possibly by placing a larger outer loop around your loop.