UNCA/NCSU ECE 209
Final Exam Fall 2011
12 December, 2011

This is a closed book and closed notes exam. It is to be turned in by 11:30 AM. ***No*** calculators, PDA's, cell phones, or other electronic or communication devices may 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 (4 points)
Complete the following code fragment so that it fills in the appropriate fields of the `name` structure `iam` with your name. I suggest you use `strcpy`.

```
struct name {
  char last[50] ;
  char first[50] ;
} ;
struct name iam ;
```

Problem 2 (3 points)
Suppose variables `x`, `y`, and `z` are declared as followed.

```
char x ;
char * const y ;
char const * z ;
```

In the following boxes are six assignment statements using these variables. Which of the following would either result in a strong warning or an error from the C compiler? Clearly state if the assignment is legal and, if not, the nature of the warning or error it would generate.

`y  = &x ;`
`z  = &x ;`
`*y = 'A' ;`
`*z = 'B' ;`
`z  = 'C' ;`
`z  = y ;`

Problem 3 (18 points)
Below is a sequence of C statements defining some variables

```
     int    whole = 10 ;
     double fract =  2.5 ;
     char  *cword = "2011" ;  /* '0' to '9' is 48 to 57*/
```

The table below, which continues on the following page, 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 character or integer, you may write the value as either an integer or character literal. That is, either `'0'`or `48` would be acceptable for the integer 48.
- If the value is a `double`, be sure to use the C syntax for `double` literals.
- If the value is a null-terminated C string (or technically, a pointer to a null-terminated C string); write your answer as a C string literal, for example `"ece 209"`.

Some of these are tricky, but none require complex arithmetic. The first two are already completed as examples of correct answers for characters and strings.

`cword[2]`	`'1'` *or* `49`
`cword+2`	`"11"`
`++whole`	
`whole++`	
`whole / fract`	
`whole * fract`	
`whole = fract`	
`fract = whole`	
`whole-- / 5`	
`--whole == 10 || whole`	
`whole += -10`	
`fract +- -5`	
`1 && whole++ ? whole : 0`	
`1 || whole++ ? whole : 0`	
`strlen(cword)`	
`strlen(&cword[2])`	
`cword[4]`	
`strstr(cword, "12") != NULL`	
`strchr(cword, '0')`	
`!isalpha(*cword)`	
`* ++ cword`	
`* cword ++`	

`5 / 3`	
`5 % 3`	
`5.0 / 3`	
`5 / 3 * 5.0`	
`5 + 4 * 2`	
`5 * 4 + 2`	
`4 * 2 / 4`	
`4 / 2 * 4`	
`(int) 3.2 + 2`	
`5.2 > 5`	
`5.2 < 5.0`	
`(2, 3) + 5`	
`6 > 5 > 4`	
`4 && 5`	
`5 == (4 || 5)`	
`! 5`	
`1 && 0 + 1`	
`1 + 0 && 1`	
`(5 || 7) && ! (5 || 7)`	
`'5' - '3'`	

Problem 4 (3 points)
Suppose `x` has been declared as a `double` variable.

What `printf` call would you use to print `x` within a field of 9 characters to a precision of 3 decimal places? Assume that `x` is less than 100000.

What `printf` call would you use to print `x` as a percentage rounded to the nearest whole percentage point? Assume that `x` is between 0 and 1. You will need to do a little arithmetic with your `printf`. For example, if `x` is 0.785398163, your statement will need to print `x` as `79%`.

Problem 5 (12 points)
Each of the following six `for` or `while` loops, which are sometimes preceded by a few initialization statements, print numbers or characters. For each loop write in the four boxes below each loop the first four lines printed by that loop. If the loop prints less than four lines, fill in a box for each line that is printed.

```
int i = 5 ;
int j = 6 ;
while (i != j) {
  i = j ;
  j = i ;
  printf("%d\n", i) ;
}
```

```
int i ;
for (i=0; i<2 || ++i<5; ++i){
  printf("%d\n", i) ;
}
```

```
int i = 10 ;
while (i > 4) {
  i /= 2 ;
  printf("%d\n", i) ;
}
```

```
int   i ;
char *c = "ECE209" ;
for (i=0; c[i]!='\0'; i+=2) {
  printf("%c\n", c[i]) ;
}
```

```
int i = -3 ;
while (i+1) {
  printf("%d\n", ++i) ;
}
```

```
int i ;
for(i=0; i<4; i=i*i) {
  printf("%d\n", i) ;
}
```

Problem 6 (12 points)
For the following blocks of C code, draw the pointers as they exist when the blocks are executed and write the output that will be printed as the code is executed. This would be a good place to explain your work.

```
int *p ;
int *q ;
int  v[3] ;
v[0] = 222 ;
v[1] = 333 ;
v[2] = 555 ;
printf("%d %d %d\n", v[0], v[1], v[2]) ;

p = &v[0] ;
v[0] = *(p++) ;
v[1] = (*p)++ ;
v[2] = *p+*p;
printf("%d %d %d\n", v[0], v[1], v[2]) ;
```

```
p = &v[1] ;
q = p+1 ;
*p = *p + 2 ;
*q = *q + 3 ;
printf("%d %d %d\n", v[0], v[1], v[2]) ;
```

```
p = q = &v[0] ;
*p = *p + 2 ;
*q = *q + 3 ;
printf("%d %d %d\n", v[0], v[1], v[2]) ;
```

```
p = &v[0] ;
q = p ;
++*++p ;
printf("%d %d %d\n", v[0], v[1], v[2]) ;
```

```
++*p++ ;
*p = p-q ;
printf("%d %d %d\n", v[0], v[1], v[2]) ;
```

The remaining problems are based on homeworks E4, P8, P9, and P12. Homework E4 was covered in Exam 1, and homeworks P8 and P9, in Exam 2; so some of these problems may seem a little familiar.

Write your answers to these problems on sheets of lined paper provided by the instructor.

Problem 7 (12 points)
For this problem, assume you have been given a C function called `median` that returns the median (middle) value of an array of `double`s and confirms to the prototype shown below:

```
    double median(const double V[], int N) ;
```

where `V` is the array of `double`s and `N` is the number of elements that `V` contains.

Subproblem 7A
Write C statements to read a series of decimal numbers and store them into an array of `double`s. The input to your program will start with a single integer which specifies the number of decimal numbers to be stored in the array. The decimal numbers will follow. After you read the first number, call `calloc` to allocate space to store the array of `double`s.

All the numbers are separated by white space. Here is some example input for your program. It is for an array of five numbers with a median value of 13.3768:

```
   5   10.5    13.3768
      3.1416  2011   17.3
```

Your program should store the size of the array into a variable called `size` and store a pointer to your allocated array into a variable called `values`. These are declared below:

```
  int     size ;
  double *values ;
```

Subproblem 7B
In this subproblem, write the *single* C statement needed to call `median` with `size` and `values` and store the value it returns in the variable declared below:

```
  double medValue ;
```

Subproblem 7C
Let’s end with a minor modification of Homework H4. Compute and print two averages: (1) the average of the numbers within the array `values` that are less than `medValue` and (2) the average of the numbers within the array `values` that are greater than `medValue`.

You may assume that the input contains at least two distinct numbers to avoid division by zero.

Problem 8 (12 points)

In this problem you are going to write code that examines a two-dimensional grid of characters to see if it *could* contain a crosshair. For the purposes of this question, a crosshair is composed of

1) a row that contains a single plus sign, `+`, where all other characters of the row are hyphens, `-`; and
2) a column that contains a single plus sign, `+`, where all other characters of the column are vertical bars, `|`.

The diagram below represents an 8×16 array with a single crosshair, shown in bold face and highlighted with a light gray fill.

```
++++++++|+++++++
+ \/ece | \\/  +
+ /\    | //\ \/
+   \ \/|     /\
+ \/  /\| / hi +
--------+-------
+ /    /|\     +
++++++++|+++++++
```

This formal definition prevents the grid from having more than one crosshair. The definition also allows the center of the crosshair, the +, to be on an edge or even a corner of the array.

In this problem we'll consider the grid to be a two dimensional array of `ROWS`×`COLS` characters. Assume that `ROWS` and `COLS` have been given values by `#define`'s. Use this declaration in the problem:

```
  char probGrid[ROWS][COLS] ;
```

If the grid has a crosshair; it has only one *and* the row of that one crosshair will be the only row in the grid with one plus sign and `COLS-1` hyphens. You're not going to solve the entire problem here. Instead, write a section of C code that looks for a row with one plus sign and `COLS-1` hyphens. (There could be more than one row like this; but, if there is more than one, the grid cannot contain a crosshair.) Your program should examine `probGrid` and set three `int` variables as directed in the following table:

`possibleRows`	Set to the number of rows within `probGrid` that contain one plus sign and `COLS-1` hyphens.
`gridRow`	If `possibleRows` is 1, contains the index of *the* row of `probGrid` with one plus sign and `COLS-1` hyphens. Otherwise, the value of `gridRow` is undefined.
`gridColumn`	If `possibleRows` is 1, contains the index of the column within row `gridRow` of `probGrid` where the plus sign is located. Otherwise, the value of `gridColumn` is undefined.

For the example grid shown above, `possibleRows` would be 1, `gridRow` would be 5, and `gridColumn` would be 8. You may assume these three variables have already been declared.

You could finish up the problem by checking the column to make sure all its other characters are vertical bars. I'm sure you'll enjoy doing that over the semester break.

Problem 9 (12 points)
In this problem you are to use the C character, string, and file functions in your solutions. These functions are described in the distributed C Reference Card.

The following declaration will be used in the various subproblems.

```
  char s[100] ;                /* a string */
```

You **must** assume that `s` is a null-terminated string in answering these questions.

Be sure to include appropriate declarations and `#include`'s in your answers.

Write short sections of C code to solve the following two subproblems.

Subproblem 9A
Modify `s` by removing any leading and trailing whitespace. This is often called "trimming" a string. For example, if `s` originally contains "`   a little  off  the sides  `", it should be transformed to "`a little  off  the sides`".

Subproblem 9B
If `s` contains any substring "`Charlotte`" followed by a punctuation mark, replace "`Charlotte`" with "`Asheville`". For example, suppose `s` originally contained:

```
   Charlotte Parker is going to Charlotte, NC, to visit UNC Charlotte.
```

Then it should be replaced with

```
   Charlotte Parker is going to Asheville, NC, to visit UNC Asheville.
```

You will *not* receive full credit for this subproblem if you use nine assignment statements to replace each character of "`Charlotte`" with the corresponding character of "`Asheville`" or if you use nine separate comparison operators to find "`Charlotte`" within the string. That is just too hard to read.

Problem 10 (12 points)
In homework P12, you used the following C `typedef` and `struct` definitions to represent a sparse matrix.

```
typedef struct sparceElementRec *sparceElementPtr ;

/* The elements of the sparse matrix */
struct sparceElementRec {
  int    row ;                      /* Position in the matrix  */
  int    col ;
  int    value ;                    /* Value of element  */
  sparceElementPtr   nextOnRow ;    /* Pointers to other
  sparceElementPtr   nextOnCol ;       other elements */
} ;

struct sparceMatrix {
  int    numRows ;                  /* number of rows */
  int    numCols ;                  /* number of columns */
  sparceElementPtr   *rows ;        /* rows */
  sparceElementPtr   *cols ;        /* columns */
} ;
```

Subproblem 10A
Write a function conforming to the following C prototype that dynamically allocates a `struct spaceElementRec` and returns a pointer to it.

```
   sparceElementPtr allocateElement(int row, int col) ;
```

The `row` and `col` fields of the returned structures should be set to the `row` and `col` values passed to the function. The `value` field should be set to 0. The `nextOnRow` and `nextOnCol` fields should be set to `NULL`. Your program should return `NULL` if it is unable to allocate space for the element structure.

Subproblem 10B
In homework P12, the sparse matrix was represented by a pointer to a structure of type `struct sparceMatrix`. In this structure, the field `numRows` contains the number of rows in the sparse matrix. Another field of the structure is the array `rows` which has `numRows` entries. The address of the first element for the linked list storing the *n*'th row is contained in `rows[`*n*`]`. If the *n*'th row has no elements, `rows[`*n*`]` is `NULL`.

In this subproblem, your task is to multiple the matrix by an integer scalar `k`, that is, to multiply each element of the sparse matrix by `k`. Do this by writing a function conforming to the following C prototype:

```
 void multiplySparceMatrix(struct sparceMatrix *M, int k) ;
```

From writing `incrementValueSparceMatrix`, you know how to change values of the sparse matrix and. From writing `printRowSparceMatrix`, you know how to go through all the values of an row. Use this experience to complete this subproblem.