Name (PRINTED): _____

University ID #: _____

Section # (or lab time) _____

CMSC 106 Exam #2 Fall 2007

Do not open this exam until you are told. Read these instructions and Sign where it asks you to acknowledge the date, time and location of our final exam.:

This is a closed book, closed notes exam. **No calculators or other aids are allowed.** If you have a question during the exam, please raise your hand. Each question's value is written next to its number. If you need more space to answer any question, please raise your hand and we will provide you with the necessary paper. You must clearly indicate on that paper which question it continues and clearly indicate on your exam paper which question is continued. You must write your name on that paper and submit it with your exam paper (even if you later decide you didn't need the extra space).

I understand that the CMSC 106 Final Exam will be on Monday, Dec 17 from 10:30am to 12:30pm in CSIC 2117.

Signature _____

There are no infinite loops or syntax or execution errors in any of the C code on this exam, except possibly where specifically noted in a problem. If you don't recognize a C construct, do the best you can.

Please make sure you have and do:
8 pages (pages numbered 1-8), 5 problems, 130 points, 70 minutes.

PLEASE WRITE LEGIBLY. Credit cannot be given for any parts of answers which cannot be read. Answers written on scratch paper will not be graded unless clearly marked.

Good luck!

2	3	4	5	6	7	8	Total

Additional scratch paper is provided a the end of the exam.
Any answers written there must be clearly labeled as to what you are answering there, // and it must be clearly labeled at the place it is continued from.
(nothing written on this page will be graded)

1. [31 pts.] Short answer:

   a. What character can be placed before any variable's name in order to find the address of where that variable is stored in memory?

   ____&____

   b. What character can be placed before any variable's name in order to dereference the value stored in that variable?

   ____*____

   c. Put "0" before any of the following statements which are false and "1" before any of the following statements which are true (be sure to mark every statement).

      i. ___0___ Not having a prototype for every function will automatically cause a syntax error during compilation.

      ii. ___0___ If the names for the parameters given in the prototype do not match the names for the parameters given in the function definition, this is a syntax error.

      iii. ___1___ Variables defined inside a pair of curly braces have "Block Scope".

      iv. ___0___ Variables defined inside a pair of curly braces will always cease to exist when the block ends and be recreated if this block were to run a second time.

      v. ___0___ When a function has no parameters, you should put the word NULL between the parenthesis to indicate it is an empty list.

      vi. ___0___ Dereferencing a pointer which has not yet been set to point to anything will usually cause a compilation error because it needs to be set to some address before it can be dereferenced.

      vii. ___0___ Referencing element 5 of a 4 element array will always cause a compilation error.

      viii. ___0___ Referencing element 5 of a 4 element array will always cause an error that terminates the run of the program immediately.

      ix. ___0___ When a function is defined but no return type is specified (when it is omitted), it is assumed to be a void function.

      x. ___0___ `while (!EOF)` — will work to test for end of file.

   c. What is the purpose of the comma operator? (For example, if you have the statement `x = 5,4;` what value is assigned to x?)

   ```
   It returns the value of the second operand.
   (i.e. x gets the vlue 5)
   ```

   d. Compare and contrast "break" and "continue" as they relate to loops. This means give at least one similarity and at least one difference.

   ```
   Both will terminate the curent iteration of the current loop.
   break will prevent further iterations of the loop.
   continue goes onto determine if further iterations are needed.
   ```

2. [30 pts.] Give the complete output of each of the following C programs. You need not be concerned with showing spaces printed, only with showing the correct values printed on the correct lines.

a.
```
#include <stdio.h>
#define SIZE1 4

main(){
  int i = 1;
  int arr[SIZE1] = {2, 1, 5, 3};
  arr[0] = arr[i+2];
  arr[3] = arr[i] + 1;
  arr[1] += 1;
  i = SIZE1;
  for (i--; i >= 0; i--)
        printf("%d/",arr[i]);
  printf("\n");
  return 0;
}
```

2/5/2/3/

b.
```
#include <stdio.h>
#define SIZE1 3

main(){
  int i = 1;
  int a[SIZE1] = {2, 3}, b[SIZE1];
  for (i=0; i < SIZE1; i++)
        printf("%d ",a[i]);
  printf("\n");
  for (i = 0; i <SIZE1; i++)
        b[i] = a[SIZE1-i-1];
  for (i = SIZE1; i > 0; i--)
         printf("%d %d\n", a[i-1],b[i-1]);
  return 0;
}
```

2 3 0

0 2

3 3

2 0

3. [15 pts.] Write a single function that will prompt for and read input from the user. The program should request one floating point value, assume the user has typed exactly one floating point value, and then read one floating point value from the user before requesting another. The function will stop requesting and reading input when the user types a 0 as the value. The function will then return a count of how many large values were given and a count of how many small values were given. Large values are those that are greater than 100 and small values are any values that are less than or equal to 100 (this also includes the negative values). Both of these must be returned to the caller through the parameter list. The zero value that was used as a sentinel should not be counted at all.

```
void get vals(int *bcnt, int *scnt){
  float curval;
  int done = 0, bc = 0, sc = 0;

  do{
    printf("give a value: ");
    scanf("%f", &curVal);
    if(curVal == 0){
      done = 1;
    } else if (curVale> 100){
      bc++;
    } else {
      sc++;
    }
  } while(!done);

  *bcnt = bc;
  *scnt = sc;
  return;
}
```

4. [29 pts.] Give the complete output of each of the following C programs. You need not be concerned with showing spaces printed, only with showing the correct values printed on the correct lines.

a.
```
#include <stdio.h>
int funct2(int b,int a);
main(){
  int a = 5, b = 6, c[2]={3,1};
  printf("%d %d %d %d\n",a,b,c[0],c[1]);
  funct2(c[0],c[1]);
  printf("%d %d %d %d\n",a,b,c[0],c[1]);
  b = funct2(a,a);
  a = funct2(b,c[0]);
  printf("%d %d %d %d\n",a,b,c[0],c[1]);
  return 0;
}
int funct2(int b, int a){
        int ans = 0;
        b = 2 * a++;
        a = a + 10;
        if (a + b > 25) ans = 1;
        return ans;
}
```

5 6 3 1

5 6 3 1

0 1 3 1

b.
```
#include <stdio.h>
#define SIZE1 3
#define SIZE2 2

int funct3(int a[],int sz);
main(){
  int i = 1, j = 0;
  int arr[SIZE1] = {5,3,2};

  funct3(arr,SIZE2);
  for (i = 0; i < SIZE1; i++)
        printf("%d ",arr[i]);
  printf("\n");
  return 0;
}
int funct3(int a[],int sz){
    int i, arr[SIZE1];
    for (i = 0; i < sz; i++)
        arr[i] = a[i]++ * 10;

    for (i = 0; i < sz; i++)
       printf("%d %d\n",arr[i], a[i]);
    printf("***\n");
    return a[0];
}
```

50 6

30 4

***

6 4 2

5. [25 pts.]

a. Give the prototype for the function named compareArrays which takes in two single dimensional arrays of characters as the first two parameters (note these are not necessarily C-strings), an integer which indicates the size of both of these arrays as the third paramter and an integer as the last parameter. The function returns an integer as the return value of the function which will tell the result of the comparison of the two arrays as defined below.

```
int compareArrays(char a[], char b[], int sz, int dir);
```

b. Implement the function. The last parameter indicates the order in which the comparison should happen: if it is a 0, the function should compare them in the same order (comparing the first elements of each and then comparing the second elements of each) but if the last parameter is a 1, the function should compare by reversing one (comparing the first element of one to the last element of the other and the second elemment of one to the second last element of the other) etc. You may assume that this last parameter is given either the value 1 or the value 0. You may also assume the arrays are the same size and that this size is accurately reflected by the value of that third parameter. The function then returns the number of differences found when they are compared in the order specified. For example if a,b,c is compared in forward (0) order to c,b,a there are two differences (a/c and c/a) but if they are compared in reverse (1) order there are no differences. A second example is if a,b,c,d is compared in forward order to d,b,c,a there are two differences (the a/d and the d/a) and if it is compared in reverse order there are also two differences (the b/c and c/b).

```
int compareArrays(char a[], char b[], int sz, int dir){
  int diffCnt = 0;
  int i = 0;

  for(i = 0; i < sz; i++){
    if(dir == 0 && a[i] != b[i])
      diffCnt++;
    else if(dirr == 1 && a[i] != b[sz-1-i])
      diffCnt++;
  }

  return diffCnt;
}
```

c. Write the code call the function you implemented on the previous page. You can assume the variables shown here already exist as follows. Any other variables you need must be declared at the beginning of the portion you write.

```
#include <stdio.h>
#define SIZE 20
/* assume the prototype above would go here */
int main(){
    char arr1[SIZE] = {'a','b','c'};
    char arr2[SIZE] = {'c','b','a'};
    int sizeUsed = 3;
/* or the following are defined (assume you don't know which
    char arr1[SIZE] = {'a','b','c','d'};
    char arr2[SIZE] = {'d','b','c','a'};
    int sizeUsed = 4;
*/
```

The function is called in such a way that it prints the most appropriate of the three statements:

- "more similar in forward order"
- "more similar in reverse order"
- "same either way"

This means that the function needs to be called comparing the two arrays (arr1 and arr2) in both forward and in reverse order (as defined above) and then needs to determine which comparison found the fewest number of differences in order to select the correct statement.

```
  int fdiff, rdiff;

  fdiff = compareArray(arr1, arr2, sizeUsed, 0);
  rdiff = compareArray(arr1, arr2, sizeUsed, 1);

  if(fdiff == rdiff)
    printf("same either way\n");
  else if(fdiff < rdiff)
    printf("more similar in forward\n");
  else
    printf("more similar in reverse\n");

  return 0;
}
```