CMSC 313 Spring 2010
Midterm Exam 1
Section 01
March 3, 2010

Name_____ Score _____

UMBC Username _____

Notes:

a. Please write clearly. Unreadable answers receive no credit.
b. For short answer questions your answer should be short, clear and to the point, not long and rambling.
c. For True / False question, write the word TRUE or FALSE; do not use the letters T and F. ***Using T and F will result in a 2-point deduction***.
d. There are no intentional syntax errors in any code provided with this exam. If you think you see an error that would affect your answer, please bring it to my attention.
e. You may assume that any necessary .h files have been #included where necessary.

True / False (2 points each)

1. _____ Both **`scanf( )`** **`and fscanf( )`** can be used to read user input from the standard input.

2. _____ In C, "false" is always equivalent to 0 and "true" is always equivalent to 1.

3. _____ In C, functions are uniquely identified by their name, so no two .c files may have functions with the same name.

4. _____ The string **"Frodo LIVES!"** requires 12 bytes of memory.

5. _____ In C, all functions have global scope.

6. _____ Other than the function prototypes, there is no difference between **`malloc( )`** and **`calloc( ).`**

7. _____ If **`p`** is a pointer to integers and **`A`** is an array of integers, then the statement **`*p = A[0];`** assigns the value stored in **`A[0]`** to **`p`**.

8. _____ Given the declaration **`char name[ 10 ] = "Bob",`** the value retuned from **`strlen( name )`** is 3.

9. _____ A function prototype tells the compiler the type of the value that is returned by the function.

10. _____ **`typedef`** is used to give a new name to an existing data type

Multiple Choice (2 points each)

(Write the letter of the best answer in the space provided)

11. _____ In C the length of a string is determined....

    [a.] from the declared size of the array which contains it.
    [b.] by a length byte stored at the beginning of the string.
    [c.] by a null terminator stored at the end of the string.
    [d.] by a length byte stored at the end of the string

12. _____ Let **p** be a pointer to an integer and let **n** be an int variable. Then after the assignment **p = &n;** the value of ***p** is

    [a.] the address of the pointer **p**
    [b.] the value stored in the variable **n**
    [c.] the address of the variable **n**
    [d.] none of the above

13. _____ The expression **`sizeof(struct foo)`** refers to

    [a.] the number of member variables in **`struct foo`**
    [b.] the size of the largest member variable in **`struct foo`**
    [c.] the total size of **`struct foo`** in bytes
    [d.] the number of pointer members in **`struct foo`**

14. _____ What is the advantage of using a pointer to a structure as a parameter to a function, instead of the structure itself ?

    [a.] The code is easier to read.
    [b.] It's more efficient because the structure is not copied.
    [c.] There is no difference; it is a matter of style which is used.
    [d.] Passing a structure as a parameter is not allowed.

15. _____Which of the following can **NOT** be done with a structure variable ?

    [a.] assign from one struct to another using "="
    [b.] pass as an argument to a function
    [c.] return as the value of a function
    [d.] all of the above can be done

16. _____ When an index is used that is too large for its array

[a.] The compiler displays a warning
[b.] The compiler displays an error
[c.] Your program usually terminates with a segfault
[d.] The C runtime environment throws an exception

17. _____ Given the declarations **`int A[20], *p, x`**; which of the following statements is ***NOT TRUE***

[a.] **`p = A;`** is equivalent to **`p = &A[0];`**
[b.] **`x = p[ 4 ];`** is equivalent to **`x = *(p + 4);`**
[c.] **`x = p[ 3 ];`** is equivalent to **`x = *(A + 3);`**
[d.] **`x = *A + 7`**; is equivalent to **`x = A[ 7 ];`**

18. _____What is the output of the following code fragment?

```
int n = 9;
int *p;
p = &n;
n++;
printf ("%d, %d", *p + 2, n);
```

[a.] 11, 9
[b.] 9, 10
[c.] 12, 10
[d.] 11, 10

19. _____ If **`endpoint`** is a variable of type **`struct coordinate,`** and **`x_coord`** is a member **`of struct coordinate`**, and **`p`** points to **`endpoint`** then to access the **`x_coord`** member of **`endpoint`** using **`p`**, use the syntax

[a.] **`p->x_coord`**
[b.] **`p.x_coord`**
[c.] **`*(p.x_coord)`**
[d.] **`p->endpoint.x_coord`**

20. _____Given the declaration **`char string[ ] = "abcdefg";`** what is output by **`printf ("%s\n", string + 3);`**

[a.] abcdefg + 3
[b.] abc
[c.] defg
[d.] the statement is illegal

21. _____ When an array parameter is passed to a function

[a.] the elements of the actual array are copied for the function to use
[b.] the function parameter is a pointer that holds the address of the array
[c.] the programmer must write code which allocates enough space for the function to store the array.
[d.] a compiler warning is displayed

22. _____ Given the declarations **`int x = 7, *p = &x;`** then the statement **`p = p + 1;`**

[a.] adds 1 to p
[b.] adds 7 to p
[c.] adds the sizeof (int) to p
[d.] adds 7 * sizeof(int) to p

23. _____ When a function exits, the values stored in the function's automatic (i.e. not static) local variables are

[a.] saved until the next time the function is called
[b.] reset to 0
[c.] lost forever
[d.]. none of the above

24. _____ Given the declarations **`int x = 7, *px = &x, **ppx = &px`**; which of the following is ***NOT TRUE***

[a.] The value of **`*px`** is 7
[b.] The value of **`**ppx`** is 7
[c.] The value of **`*ppx`** is the address of **`px`**
[d.] all of the above are true

25. _____ Which of the following is ***NOT TRUE***

[a.] **`malloc( )`** and **`calloc( )`** both allocated memory from the heap
[b.] **`free( )`** is used to return memory to the heap
[c.] **`malloc( )`** and **`calloc( )`** both return NULL if allocation fails
[d.] memory obtained from **`malloc( )`** and **`calloc( )`** is uninitialized

The next 5 questions refer to the following declarations:

```
typedef struct person
      {
         char ssn[9];
         int   age;
         char  *name;
      } PERSON;
```

```
typedef struct student
   {
         PERSON p;
         char nickname[20];
         int  weight;
   } STUDENT;
```

```
STUDENT s1, *pStudent = &s1;
PERSON  p1, *pPerson1 = &p1;
PERSON  p2, *pPerson2 = &p2;
```

26. _____ The expression **s1.p.ssn[3]**
    [a.] is a char
    [b.] is an array of char
    [c.] is a pointer to char
    [d.] is illegal and causes a compiler error

27. _____The expression **pStudent->weight**
    [a.] is an int
    [b.] is equivalent to **s1.weight**
    [c.] both a and b
    [d.] is illegal and causes a compiler error

28. _____The expression **s1.p->age**
    [a.] is an int
    [b.] is equivalent to **pStudent->p->age**
    [c.] both a and b
    [d.] is illegal and causes a compiler error

29. _____The statement **p1 = p2;**
    [a.] is a structure assignment statement
    [b.] has the same effect as ***pPerson1 = *pPerson2;**
    [c.] both a and b
    [d.] is illegal and causes a compiler error

30. _____The statement **pPerson1 = pPerson2**
    [a.] is a structure assignment statement
    [b.] causes **pPerson1** to point to **p2**
    [c.] both a and b
    [d.] is illegal and causes a compiler error

Short Answer

31. **(4 points)** If not used carefully, C library functions such as **`strcpy( )`** and **`strcat( )`** that manipulate strings can lead to subtle errors or program termination (segmentation fault). In two sentences or less, explain why this is so.

_____

_____

32. **(4 points)** In no more than two (2) sentences, explain why it is necessary to use the C string library function **`strcmp( )`** rather than using the equality operator == to compare two strings.

_____

_____

33. **(5 points)** In no more than two (2) sentences, give a brief description of the task this function performs on the lines provided. DO NOT describe each line of code or the individual "steps".

```
void mystery( double *dp, int n)
 {
    int k, N = n - 1;
    for (k = 0; k < n / 2; --N, ++k)
    {
         double temp = *(dp + N);
          *(dp + N) = *(dp + k);
          *(dp + k) = temp;
    }
 }
```

_____

_____

34. **(4 points)** Examine the code in the boxes, then answer the questions below

```
/** main.c **/

extern int randomInt;
void getRandomInt( int x );

static void printRandoms( int n )
{
  int k;
  for (k = 0; k < n; k++)
  {
     getRandomInt( 12345 );
     printf("%d\n", randomINt);
  }
}

int main( )
{
  printRandoms( 5 );
  return 0;
}
```

```
/** random.c **/

int randomInt;

void getRandomInt( int max )
{
   static long lastRandom = 100001;

  lastRandom
     = (lastRandom * 125) % 2796203;

  randomInt = (lastRandom % max) + 1;
}
```

a. What is the **scope** of `randomInt`?

b. What is the **lifetime** of `randomInt`?

c. What is the **scope** of `lastRandom`?

d. What is the **lifetime** of `lastRandom`?

35. **(4 points)** Some languages (like Java) support a Boolean data type (e.g bool) that can hold the values "true" and "false" which are often keywords of the language. C is NOT one of those languages, but a "clever" C programmer can simulate the bool type as well as true and false. Explain how you might use typedef(s) and / or #define(s) to simulate the **`bool`** type, and the values **`true`**, and **`false`** in C.

36. **(4 points)** Examine the code below, then answer the questions that follow.

```
int i;
int *grades[5];

for (i = 0; i < 5; i++)
   grades[i] = malloc( (i + 1) * sizeof(int));
```

a. What is the data type of **`grades`**? _____

b. What is the data type of **`grades[ 1 ]`** _____

c. What is the data type of **`grades[4][2]`** ? _____

d. How many ints can be stored in **`grades`**? _____

37. **(5 points)** Fill in the blanks for the function named GetXY( ) below that prompts the user for the x- and y-coordinates of a point and "returns" the coordinates via function parameters. The x- and y-coordinates are integers.

```
_________ GetXY(___________, ____________)
{
  printf( "Input the x-coordinate: ");

  scanf("%d", ____________);

  printf( "Input the y-coordinate: ");

  scanf("%d", ____________);
}
```

38. **(10 points)** Fill in the blanks in the function below. This function returns the largest odd integer from an array of integers, all of which are supposed to be positive (but you can never be too careful). The function returns -1 if there are no odd integers in the array.

```
int LargestOdd (int a[ ], int size)
{
    int k, largest = ________;

    for (k = 0; k < size; k++)
    {
        assert( ____________________ );

        if((_________________) && (_________________))

           ____________________________;

     }

     return largest;
 }
```