Name (PRINTED): _____

Student ID #: _____

Section # (or TA's: name and time) _____

CMSC 106 Exam #1 Fall 2007

Do not open this exam until you are told. Read these instructions:

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. Watch the number of points for each question to be sure you are not spending too much time on any particular question. It is not necessary to do the problems in the order presented – the order of presentation was selected to give extra space to questions that may need that space. (In particular - the page after a programming question is blank so you can flow onto the second page if needed.)

No one may leave before the exam is over.

There are no syntax errors in any of the C code on this exam. If you don't recognize a C construct, do the best you can.

pages numbered 1-8, 6 problems, 100 points, 60 minutes.

PLEASE WRITE LEGIBLY. Credit cannot be given for any parts of answers which cannot be read.

Good luck!

1	2	3	4	5	6	Total

1. [18 pts.] Short answer:

   a. Give the UNIX command which will move you into directory where the class example source files are. You may assume you are in your home directory when you give the command.

      `cd ~/106public/ClassExamples`

   b. Write the UNIX command which will allow you to display the contents of the file named `this.file` to the screen one page at a time. The file named `this.file` is in your home directory. You do not know which directory you are in as you give the command.

      `more ~/this.file`

   c. Give the command you would type that would compile the program that is named `this.c` into an executable file named `that`. The file named `this.c` is in the current working directory and the executable file can be put in the same directory.

      `gcc -o that this.c`

   d. There are certain things that are usually at the very top of the source file.

      i. These often begin with a the `#` character. What is the purpose of the `#` character in this context?

         `the preprocessor will do this task`

      ii. One of these is `#include` - what is the purpose of this?

         `to bring in a file (usually a header file)`

      iii. Another is `#define` - what is the purpose of this?

         `to define a macro or a symbolic constant`

   e. Give an example of an escape sequence which could appear within a `printf` string and tell its purpose.

      \t tab; \n newline; %% %sign; \\ backslash

   f. Give an example of a format specifier which could appear within a `printf` string and tell its purpose.

      %d for integer; %f for floats

2. [15 points]Longer answer (paragraph - multiple sentences).

   a. **Briefly** describe a difference between a syntax error in a program and a semantic error in a program. How can you tell when you have one type or the other?

   - syntax error means you have violated the rules of te language and the compiler won't be able to translate it

   - semantic error means that the meaning of the code is not what you intended

   b. **Briefly** describe the term "short circuiting" and where it is used in C.

   it makes it so as soon as the value of an expression is known for sure, the rest of the expression is not evaluated

   with && if the left is faluse, don't evaluate right

   with —— if the left is true, don't evaluate right

   c. Describe how machine code (1st Generation Language) is different from assembly code (2nd Generation Language).

   assembly code has code words while machine code is all a sequence of bits

3. [20 pts.] Write the complete C program which will request a single integer from the user. If that integer is positive it will then say "Thank you" and then request a character, but if the integer was not positive the program will end immediately with the message "Invalid Input". Once the character is read, the program will print "Invalid Input" if the character was not a, b, c or d (either upper or lowercase should be accepted), but if the character is one of those listed it should print "Thank you". It should then print the character entered the number of time indicated by the integer. The dividing lines appear in this output just as a division between the different runs of the program. If the program did not run completely and had to end with "Invalid Input", it should return any negative value to the operating system. If the program did run completely and ended by printing the character the correct number of times, it should return a 0 to the operating system.

```
--------------------
Enter an Integer: -3
Invaid Input
--------------------
Enter an Integer: 2
Thank you
Enter a Character: x
Invalid Input
-------------------
Enter an Integer: 3
Thank You
Enter a Character: a
Thank you
aaa
--------------------
```

```
#include <stdio.h>
int main(){
  int inInteger, chCount = 0;
  char inChar, junk;
  printf("Enter an Integer:";
  scanf("%d", &inInteger);
  if(inInteger <= 0){
    printf("Invalid Input\n");
    return -1;
  }
  prtinf("Thank you\n");
  printf("Enter a character:");
  scanf("%c", &junk);
  scanf("%c", inChar);
  if(inChar != 'a' && inChar != 'b' && inChar != 'c' && inChar != 'd' &&
    inChar != 'A' && inChar != 'B' && inChar != 'C' && inChar != 'D'){
    printf("Invalid Input\n");
    return -1;
  }
  printf("Thank you");
  while(chCount < inInteger){
    printf("%c", inChar);
    chCount++;
  }
  return 0;
}
```

Page Blank to leave extra room for the programming question.

4. [15 pts.] Show all of the output for each of the following sets of printf statements. Pay careful attention to the expression evaluation and format specifiers indicated. Mark each blank space with a caret "∧" to make the actual formatting more clear. Make sure to write the correct things on the correct lines. The variables used have the following types and values before that set (a or b) begins. (Notice the changes made to the variable values in one set do not effect the other.):

```
int a=23, b=12, c=5;
float x=2.1, y=1.5678, z=3.1234;
```

a.
```
printf("%-3d%.2f%1d\n",a++,x,++b);
printf("%d%d\n",a,b);
```

23∧2.1013

2413

b.
```
printf("%.3d%5.1f",a%b,y);
printf("%2d%1.1f",(a>b?5:10),z);
```

011∧∧1.6∧53.1

5. [12 pts.] For each set of variable values given below, tell which statement will be performed (just give the number). If more than one statement would be done be sure to list all statements that would be performed in the order they would be performed. You just need to give the statement's number. If no statement would be performed, write "**none**". **Be careful, the indenting may not be correct!**

```
if (a < b)
if (c < d)
    statement1;
else
if (a <= c && a != d)
if (a == c)
    statement2;
else
    statement3;
else
if (c > d)
    statement4;
else
    statement5;
```

a. statement1 a = 2, b = 5, c = 3, d = 6

b. statement4 a = 3, b = 4, c = 2, d = 1

c. none a = 1, b = 1, c = 1, d = 1

d. statement3 a = 3, b = 4, c = 5, d = 2

6. [20 pts.] Give the output of the following loops. If a loop never executes or has no output then write "none"; if a loop is infinite show the output of the first three iterations and then write "infinite". In this output, spaces do not matter - just have the correct values on the correct lines. **Be careful - indenting may not be correct!**

a.
```
i = 0;
while (i < 10){
    printf("%d - ",i++);
    i++;
}
```

0 - 2 - 4 - 6 - 8 -

b.
```
i = 10; j=2;
while (i>j*2)
    printf("%d %d\n",j,i);
    i -= j;
```

2 10

2 10

2 10

infinite

c.
```
i = 2; j = 7; k = 2;
while  (i+k <= j){
   printf("%d %d %d\n",i, j, k);
   i++; k+=3;
}
printf("%d-%d-%d\n",i,j,k);
```

2 7 2

3-7-5

d.
```
int i = 1;
int j = 11;
while (i < j/2){
    printf("i, j\n");
    j-=3;
}
```

i, j

i, j

i, j