16.216: ECE Application Programming

Summer 2012

Exam 2
August 2, 2012

Name: _____ **ID #:** _____

For this exam, you may use only one 8.5" x 11" double-sided page of notes. All electronic devices (e.g., calculators, cellular phones, PDAs) are prohibited. If you have a cellular phone, please turn it off prior to the start of the exam to avoid distracting other students.

The exam contains 3 questions for a total of 100 points. Please answer the questions in the spaces provided. If you need additional space, use the back of the page on which the question is written and clearly indicate that you have done so.

Please read each question carefully before you answer. In particular, note that:

- Question 3 has three parts, but you are only required to complete two of the three parts.
  - You may complete all three parts for up to 10 points of extra credit. If you do so, **please clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.**
- For each part of that problem, you must complete a short program. I have written part of the program for you and provided comments to describe what each missing piece of code should do.
- You can solve each problem using only the variables that have been declared, but you may declare and use other variables if you want.
- Note that you may, if you need more space, complete each of these problems on the back of another page—just clearly note where your solution is.

You will have two hours to complete this exam.

Q1: Multiple choice	/ 20
Q2: Functions and pointers	/ 40
Q3: Loops	/ 40
TOTAL SCORE	/ 100
EXTRA CREDIT	/ 10

1. (20 points, 5 points per part) ***Multiple choice***
For each of the multiple choice questions below, clearly indicate your response by circling or underlining the choice you think best answers the question.

a. You are given the following short program:

```
int i = 10;
while (i > 0) {
   i--;
   if (i > 3)
      continue;

   else if ((i % 2) == 0)
      break;
}
```

How many iterations will this loop execute?

i. 10

ii. 8

iii. 7

iv. 3

v. An infinite number—the loop never ends

b. You are given the following function prototype:

```
double f(int a, int b);
```

In a program with integer variables `x` and `y`, and double-precision variable `d`, which of the following calls to function `f()` would not compile (i.e., which one is an incorrect call to `f()`)?

i. `f(x, y);`

ii. `x = f(y, y + 2);`

iii. `d = f(&y, &x);`

iv. `d = f(5, 7);`

v. `d = f(-3, x);`

1 (cont.)

c. Given the following code snippet:

```
int x = 1;
int i = 0;
while (i <= 3) {
   x = x * 2;
   i++;
}
```

Which of the following choices can replace the `while` loop and produce the exact same value for `x`? Assume `x` is always initialized to 1.

i. `for (i = 1; i < 4; i++)`

ii. `for (i = 0; i < 3; i++)`

iii. `for (x = 0; x <= 3; x++)`

iv. `for (i = 3; i >= 0; i--)`

v. None of the above

d. Which of the following "functions" would you find most useful? Circle all that apply *(and please don't waste too much time on this "question").*

i. `set_all_16_216_grades(100);`

ii. `set_program_status(&program3, &program4, "graded");`

iii. `print_answer_key_to_screen();`

iv.
```
set_overall_GPA(4.0);
graduate_now();
```

2. (40 points) ***Functions and pointers***

For each short program shown below, list the output exactly as it will appear on the screen. Be sure to clearly indicate spaces between characters when necessary.

You may use the available space to show your work as well as the output; just be sure to clearly mark where you show the output so that I can easily recognize your final answer.

a. (14 points)

```
void main() {
   int var1, var2, var3;
   int *pt1, *pt2, *pt3;

   pt1 = &var2;
   pt2 = &var3;
   pt3 = &var1;

   var1 = 16;
   var3 = 216;
   *pt1 = *pt3 + 7;
   pt1 = pt2;
   *pt2 = *pt3 + *pt1;

   printf("%d %d %d\n", var1, var2, var3);
   printf("%d %d %d\n", *pt1, *pt2, *pt3);
}
```

2 (cont.)

b. (12 points)

```
int mac(int v1, int v2, int v3) {
   return v1 * v2 + v3;
}

void main() {
   int r1, r2, r3;

   r1 = mac(2,2,7);
   r2 = mac(-2,3,9);
   r3 = mac(r1, r2, 7);

   printf("%d %d %d\n", r1, r2, r3);
   printf("%d\n", mac(r3,r2,r1));
   printf("%d\n", mac(r1,r2,r3));

}
```

c. (14 points)

```
int swapIfGT(int *x, int *y) {
   int temp;

   if (*x > *y) {
      temp = *x;
      *x = *y;
      *y = temp;
      return 1;
   }

   return 0;
}
void printVars(int *a, int *b) {
   if (swapIfGT(a,b) == 1)
      printf("%d > %d\n", *a, *b);
   else
      printf("%d <= %d\n", *a, *b);
}
void main() {
   int v1, v2, v3, v4;
   v1 = 5;
   v2 = 7;
   v3 = 9;
   v4 = 1;

   printVars(&v1,&v2);
   printVars(&v3,&v4);
   printVars(&v2,&v3);
   printVars(&v1,&v4);
}
```

3. (40 points, 20 per part) ***Loops***

For each part of this problem, you are given a short program to complete. **CHOOSE ANY TWO OF THE THREE PARTS** and fill in the space provided with appropriate code. **You may complete all three parts for up to 10 points of extra credit, but must clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.**

a. Repeatedly prompt a user to enter two double-precision values and then read those numbers. Your program should end when the second number entered is less than the first—at that point, print the maximum value entered. A sample run is below; user input is underlined:

```
Enter two values: 1 3
Enter two values: -0.7 1.234
Enter two values: 55 55
Enter two values: 16.216 16.217
Enter two values: 2.3 -3.7
Max value: 16.217000
```

```
void main() {
   double val1, val2;    // Input values
   double max;           // Max value—you may assume, as Visual
                         //   Studio does, that this variable
                         //   initially holds the lowest
                         //   possible negative value if it's
                         //   not explicitly initialized

   // REPEATEDLY PROMPT USER TO ENTER TWO VALUES—MAY TEST END
   //   CONDITION HERE OR SPACE BELOW, DEPENDING ON LOOP TYPE




         printf("Enter two values: ");
         scanf("%lf %lf", &val1, &val2);


         // DETERMINE IF EITHER VALUE IS NEW MAX VALUE









   // MAY TEST LOOP END CONDITION HERE



   printf("Max value: %lf\n", max);
}
```

b. Prompt for and read a series of characters and count the number of whitespace characters—spaces, tabs (`'\t'`) and newlines (`'\n'`)—in the input. Stop reading if the user enters the same non-whitespace character twice in a row. Print the total number of whitespace characters. The sample below contains 1 tab, 3 spaces, and 2 newlines (input is underlined):

```
ab   3 6 ?          (Note: tab is between 'b' and '3')
h Q
zz
Total whitespace characters: 6
```

```
void main() {
      char inChar;          // Input value
      char lastChar;        // Input value from previous iteration
      char spaceCnt;        // # whitespace characters

      // INITIALIZE VARIABLES AS NEEDED




      // REPEATEDLY (1) SAVE RESULT OF PREVIOUS ITERATION, AND
      // (2) READ NEW CHARACTER




            scanf("%c", &inChar);

            // WHITESPACE CHARACTER FOUND—INCREMENT COUNT









            // INPUT CHARACTER ISN'T WHITESPACE AND MATCHES VALUE
            //   FROM PREVIOUS ITERATION--EXIT









      printf("Total whitespace characters: %d\n", spaceCnt);
}
```

c. Complete the program below so that it reads a positive, non-zero integer, n, then prints all values between 1 and n, including n. The output should be formatted as follows:
   - The first line of output contains a single value.
   - For all other lines, use the last value on the previous line to determine the number of values to print on the following line.

See the examples below for further clarification; user input is underlined.

```
Enter n: 8
1
2
3 4
5 6 7 8
```

```
Enter n: 14
1
2
3 4
5 6 7 8
9 10 11 12 13 14
```

```
void main() {
      int n;                // Input value
      int valsPerLine;      // Values to be printed on the current line
      int numPrinted;       // Number of values printed already on the
                            // current line
      int i;                // Loop index

      // INITIALIZE VARIABLES AS NEEDED



      // Prompt for and read input values
      printf("Enter n: ");
      scanf("%d", &n);

      // LOOP TO PRINT ALL VALUES BETWEEN 1 AND n, INCLUDING n



            printf("%d ", i);

            // UPDATE numPrinted



            // CHECK numPrinted--IF LIMIT HAS BEEN REACHED, MOVE TO
            //   NEXT LINE, AND SET valsPerLine AND numPrinted
            //   APPROPRIATELY









}
```