16.216: ECE Application Programming

Practice Problems: Functions and Pointers
Solution

1. What does the following program print?

a.
```
int main() {
   float var1, var2, var3;
   float *pt1, *pt2, *pt3;

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

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

   printf("var1 = %.0f, var2 = %.0f, var3 = %.0f\n",
             var1, var2, var3);
   printf("*pt1 = %.0f, *pt2 = %.0f, *pt3 = %.0f\n",
             *pt1, *pt2, *pt3);

   return 0;
}
```

Solution:
```
var1 = 16, var2 = 23, var3 = 232
*pt1 = 232, *pt2 = 232, *pt3 = 16
```

1 (cont.) What does the following program print?

b.
```
int main() {
    int x, y, z;
    int *p;

    p = &x;
    x = 2;
    *p = x + 5;
    y = x - 4;
    *p += 5;
    p = &y;          // NOTE: There was a typo in the original
                     //   problem—the ampersand was missing
    z = *p / 2;
    (*p)++;

    printf("x = %d, y = %d, z = %d\n", x, y, z);

    return 0;
}
```

Solution:

```
x = 12, y = 4, z = 1
```

1 (cont.) What does the following program print?

c.
```
double f(double x, double y) {
   x = x * 2;
   y = y – 3;
   return x + y;
}

int main() {
   double a = 2.5;
   double b = 6.0;
   double c = -1;
   double d = 0;

   a = f(a, b);
   b = f(b, a);
   c = f(c, d);
   d = f(d, c);

   printf("%lf %lf %lf %lf\n", a, b, c, d);

   return 0;
}
```

Solution:

```
8.000000 17.000000 -5.000000 -8.000000
```

1 (cont.) What does the following program print?

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

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

   r1 = mac(2,2,7);
   r2 = mac(-3,3,9);
   r3 = mac(r1, r2, 5);

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

   return 0;
}
```

Solution:

```
11 0 5
11
5
```

1 (cont.) What does the following program print?

e.
```
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);
}

int 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);

   return 0;
}
```

Solution:

```
5 <= 7
1 > 9
1 > 7
5 <= 9
```

1 (cont.) What does the following program print?
See the next page for the solution

f.
```
int f1(int a, int b, int c) {
   a++;
   b--;
   c += 5;
   return a + b + c;
}
int f2(int *a, int b, int c) {
   (*a)++;
   b--;
   c += 5;
   return *a + b + c;
}
int f3(int a, int *b, int c) {   // NOTE: return type should
                                 //   have been int--typo
   a++;
   (*b)--;
   c += 5;
   return a + *b + c;
}
int f4(int a, int b, int *c) {   // NOTE: return type should
                                 //   have been int--typo

   a++;
   b--;
   (*c) += 5;
   return a + b + *c;
}
```

Main program on next page

1f (continued)

```
int main() {
     int x = 10;
     int y = 20;
     int z = 30;
     int r;

     r = f1(x, y, z);
     printf("%d %d %d %d\n", x, y, z, r);

     r += f2(&x, y, z);
     printf("%d %d %d %d\n", x, y, z, r);

     r += f3(x, &y, z);
     printf("%d %d %d %d\n", x, y, z, r);

     r += f4(x, y, &z);
     printf("%d %d %d %d\n", x, y, z, r);

     return 0;
  }
```

Solution:

```
     10 20 30 65
     11 20 30 130
     11 19 30 196
     11 19 35 261
```

2. Write a function that does each of the following tasks:
(NOTE: You do not have to do any error checking in these functions unless the problem explicitly specifies that you do so.)

a. Given one argument, which holds a single character, do the following:
- If the character is a lowercase letter, return the uppercase version of that letter.
  - Hint: the ASCII values of `'A'` and `'a'` are 65 and 97, respectively.
- If the character is not a lowercase letter, return the original character.

Solution: *This function is essentially the "toupper" function from <ctype.h>*

```
char setUpper(char c) {

     // Character is lowercase letter--change to uppercase
     //   by subtracting 32 ('A' – 'a' = 97 – 65 = 32)
     if ((c >= 97) && (c <= 122))
          return c – 32;

     // Note that else is redundant--function returns
     //   if previous condition is true
     else
          return c;
}
```

b. Rewrite the function in part (a) so that it does not return anything; it simply modifies the original character if necessary.

Solution: *Tweak the previous function so that:*
- *The argument is passed by address*
- *We change the value if it's a lowercase letter, and do nothing otherwise*
- *The return type is now void*

```
void setUpper(char *cPtr) {

     // Character is lowercase letter--change to uppercase
     //   by subtracting 32 ('A' – 'a' = 97 – 65 = 32)
     if ((*cPtr >= 97) && (*cPtr <= 122))
          *cPtr = *cPtr – 32;
}
```

2 (cont.)

c. Given one integer argument, `nVals`, do the following:
   - If `nVals` is less than 0, return -1.
   - Otherwise, read `nVals` different values from the command window, and return the number of those input values that are negative.

Solution:

```
int countNeg(int nVals) {
     int inval;               // Input integer
     int count = 0;           // # of negative values
     int i;                   // Loop index

     if (nVals < 0)           // nVals negative--return -1
          return -1;

     // Otherwise, read nVals different inputs, and count
     //   negative inputs
     for (i = 0; i < nVals; i++) {
          scanf("%d", &inval);
          if (inval < 0)
               count++;
     }
     return count;
}
```

2 (cont.)

d. Read three values from the command line input—two double-precision variables followed by an integer, with spaces separating the three values—and return 1 if they are successfully read, 0 otherwise. Note that all three values read in should be accessible outside the function.

Solution: *Note that arguments must be passed by address, so they are accessible outside function*

```
int readThree(double *v1, double *v2, int *v3) {
     int n;              // # values read by scanf()

     // Read inputs
     // NOTE: Since v1, v2, and v3 are already pointers,
     //   can just pass them directly to scanf()
     n = scanf("%lf %lf %d", v1, v2, v3);

     if (n < 3)
          return 0;
     else
          return 1;
}
```

e. Given an integer input, `x`, go through all positive integers from 1 to 10 and check whether `x` is divisible by each value. If so, print a message indicating that `x` is divisible by the value in question. Return the number of values between 1 and 10 that divide evenly into `x`.

Solution:

```
int checkDiv(int x) {
     int i;              // Loop index
     int count = 0;      // # values that divide into x evenly

     for (i = 1; i <= 10; i++) {

          // x is divisible by i--print message and increment
          //   count
          if ((x % i) == 0) {
               printf("%d is divisible by %d\n", x, i);
               count++;
          }
     }
     return count;
}
```