16.216: ECE Application Programming

Practice Problems: 1-D Arrays, Pointer Arithmetic, and Strings
Solution

1-D Arrays

1. What does each of the following programs print?

a.
```
int main() {
   int arr[15];
   int i;
   for (i = 0; i < 15; i++)
      arr[i] = i * 3;
   for (i = 15; i > 0; i--)
      printf("%d\n", arr[i-1]);
   return 0;
}
```

Solution:

```
42
39
36
33
30
27
24
21
18
15
12
9
6
3
0
```

1. (cont.)

```
b. int main() {
       double vals[] = {1.2, 3.5, 4, -7.8, 6.7, 8.7};
       int i;
       int n = sizeof(vals) / sizeof(double);

       printf("n = %d\n", n);
       printf("vals[1] + vals[3] = %lf\n", vals[1] + vals[3]);
       printf("vals[2] – vals[5] = %lf\n", vals[2] – vals[5]);
       printf("vals[n-3] + vals[0] = %lf\n", vals[n-3] + vals[0]);
       return 0;
   }
```

Solution:

```
n = 6
vals[1] + vals[3] = -4.300000
vals[2] - vals[5] = -4.700000
vals[n-3] + vals[0] = -6.600000
```

```
c. int main() {
       int a[8] = {1, 2, 7, 0, 4, 5, 3, 6};
       int b[8] = {16, 216, 201, 202, 2011};
       int i;
       for (i = 0; i < 8; i++)
         printf("%d %d\n", a[i], b[a[i]]);
       return 0;
   }
```

Solution:

```
1 216
2 201
7 0
0 16
4 2011
5 0
3 202
6 0
```

1 (cont

```
d. void f(int arr[], int n) {
     int i;
     for (i = 0; i < n; i++)
       arr[i] = arr[n-i-1];
   }

   void printArray(int arr[], int n) {
     int i;
     for (i = 0; i < n; i++) {
       printf("%d ", arr[i]);
     }
     printf("\n");
   }

   int main() {
     int i;
     int q[6] = {1, 1, 2, 3, 5, 8};

     printArray(q, 6);

     f(q, 6);
     printArray(q, 6);

     f(q, 4);
     printArray(q, 6);

     return 0;
   }
```

<u>Solution:</u>

```
1 1 2 3 5 8
8 5 3 3 5 8
3 3 3 3 5 8
```

1 (cont.)

```
e. int f(int x[]) {
        int t = 0;
        int i = 0;
        while (x[i] != -1) {
          t += x[i];
          i++;
        }
        return t / i;
   }

   int main() {
        int a1[] = {1, 2, 3, 4, 5, -1};
        int a2[] = {-2, -4, -5, -1, 3, 2, 1};
        int a3[] = {10, -1, 20, -1, 30, -1, 40, -1};
        printf("%d %d %d\n", f(a1), f(a2), f(a3));
        return 0;
   }
```

Solution:

```
3 -3 10
```

2. Write a function to do each of the following tasks:

a. `checkIfSorted()`: Given an array of integer values, `a[]`, and the size of the array, `n`, check if the array is sorted from smallest to largest value. If so, return 1; if not, return 0.

Solution:

```
int checkIfSorted(int a[], int n) {
     int i;                 // Loop variable

     // Go through array and check if all pairs of consecutive
     //   values are sorted correctly--if so, then whole array
     //   is sorted correctly
     for (i = 0; i < n-1; i++) {

          if (a[i] > a[i+1])       // Array is not sorted
               return 0;
     }

     return 1;              // If function reaches this point,
                            //   it didn't exit the loop
                            // Array must be sorted
}
```

2 (cont.)

b. `countDiv()`: Given an array of integer values, `a[]`, the size of the array, `n`, and a value `v`, count and return the number of values in `a[]` that are divisible by `v`.

Solution:

```
int countDiv(int a[], int n, int v) {
     int count = 0;      // # of divisible values
     int i;              // Loop variable

     // Access each array element and check if divisible
     //   by v; if so, increment count
     for (i = 0; i < n; i++) {
          if ((a[i] % v) == 0)
               count++;
     }

     return count;
}
```

c. `fillArray()`: Given an empty array of double-precision values, `d[]`, and the total size of the array, `n`, repeatedly read values from the standard input and store them in `d[]` until one of two conditions occurs:

- The user enters the value 0 (which should be stored in the array).
- The array is completely full.

Once done, your function should return the number of values actually stored in the array.

Solution:

```
int fillArray(double d[], int n) {
     int count = 0;      // # of values stored in array

     // Repeatedly prompt user to enter value and store in array
     // Exit if (a) user enters 0, or (b) array is full
     do {
          printf("Enter value: ");
          scanf("%lf", &d[count]);
          count++;
     } while ((d[count-1] != 0) && (count < n));

     return count;
}
```

Pointer arithmetic

3. What does each of the following programs print?

a.
```
int main() {
   int i;
   int arr[10];
   int *p = arr;
   for (i = 0; i < 10; i++) {
      *p = i * i;
      p++;
   }
   for (i = 0; i < 10; i++)
      printf("%d\n", arr[i]);
   return 0;
}
```

Solution:

```
0
1
4
9
16
25
36
49
64
81
```

3 (cont.)

b.
```
int main() {
     double *d;
     double p[]={49.1, 90.4, 76.6, 85.3, 78.4, 80.2, 70.0};
     d = p + 2;

     printf("%lf\n", *d);
     d--;
     printf("%lf\n", *d);
     d += 4;
     printf("%lf\n", *d);
     d – 2;
     printf("%lf\n", *d);
     return 0;
}
```

Solution:

```
76.600000
90.400000
80.200000
80.200000
```

c.
```
int main() {
     int i;
     char str[] = "ece application programming";
     char *sp = str;
     for (i = 0; i < 14; i++) {
          (*sp) -= 32;      // Convert character that sp points to
                            //   to uppercase
          sp += 2;
     }
     printf(str);
     return 0;
}
```

Solution:

```
EcE ApPlIcAtIoN PrOgRaMmInG
```

4. Write code to implement your own version of each of the following string functions, using pointers to deal with each string:
   - `int strlen(char *s);`

Solution:

```
int strlen(char *s) {
     char *p;        // Pointer to check all characters
     int count = 0; // # characters checked so far

     p = s;          // Point to first character in string

     // Keep counting characters until you reach null
     while (*p != '\0') {
          count++;
          p++;
     }

     return count;
}
```

   - `char *strcpy(char *dest, char *source);`

Solution: *Note: solution assumes that there is enough room in dest to hold source, and therefore does no error checking. Also, a solution that just uses array indexing to copy from one to the other is probably more efficient, but I wanted to show you the pointer-based solution.*

```
char *strcpy(char *dest, char *source) {
     char *dPtr, *sPtr;
     dPtr = dest;
     sPtr = source;

     while (*sPtr != '\0') {        // Copy until you hit null
          *dPtr = *sPtr;
          dPtr++;
          sPtr++;
     }

     *dPtr = '\0';                  // Must add null to end
}
```

4 (cont.)

- `int strncmp(char *s1, char *s2, int n);`

Solution:

```
int strncmp(char *s1, char *s2, int n) {
     char *sPtr1, *sPtr2;
     int i = 0;

     sPtr1 = s1;
     sPtr2 = s2;

     while (i < n) {

          if (*sPtr1 < *sPtr2)          // s1 "<" s2
               return -1;

          else if (*sPtr2 > *sPtr2)     // s1 ">" s2
               return 1;

          else if (*sPtr1 == '\0')      // Both strings equal
               return 0;

          i++;      // Implied else--no conditions above true,
          sPtr1++;  //   so increment counter and move pointers
          sPtr2++;
     }

     return 0;      // First n characters must have been equal
                    //   if you reached this point
}
```

Strings

5. What does each of the following programs print?

a.
```
int main() {
   int i;
   char str[] = "1234567890abcdefghij";
   for (i = 1; i < strlen(str); i *= 2) {
      printf("%c\n", str[i]);
   }
   return 0;
}
```

Solution:

```
2
3
5
9
g
```

b.
```
int main() {
   char s1[] = "String1";
   char s2[] = "String2";
   int i;
   for (i = 1; i < strlen(s1); i++) {
      if (strncmp(s1, s2, i) == 0)
         printf("Match\n");
      else {
         printf("No match\n");
         break;
      }
   }
   return 0;
}
```

Solution:

```
Match
Match
Match
Match
Match
Match
```

5 (cont.)

c.
```
int main() {
     char s1[20] = "";
     char s2[20] = "";
     strcat(s1, "ab");
     strcat(s2, "ac");
     strcat(s1, s2);
     strcat(s2, s1);
     strncat(s1, s2, 3);
     strncat(s2, s1, 3);
     printf("%s %s\n", s1, s2);
     return 0;
}
```

Solution:

```
abacaca acabacaba
```

6. Write a function to do each of the following tasks:

a. `buildString()`: Given a character array, `str[]`, and the length of the array, `n`, repeatedly read strings from the standard input and store them in `str[]`, ensuring there is a single space between each string, until either the user enters Ctrl-Z (end of file) or `str[]` does not have enough remaining room to hold the next string. For example, if n = 10:

- User enters "one word" → str holds "one word"
- User enters "three words" → str holds "three" (not enough space to hold both words)

The function should return the actual length of the string stored in `str[]`

Solution:

```
int buildString(char str[], int n) {
     char temp[50];            // Assume 50 is max # of chars

     do {
          scanf("%s", temp);

          // Add string to end of str if there's room for
          //   that string and space
          if ((strlen(str) + strlen(temp) + 1) < n) {
               strcat(str, " ");
               strcat(str, temp);
          }
          else
               break;
     } while (1);

     return strlen(str);
}
```

6 (cont.)

b. `longestMatch()`: Given two strings, s1 and s2, return the length of the longest matching character sequence between the two, starting with the first character of each. For example:

- s1 = “string”, s2 = “other” → function returns 0
- s1 = “string”, s2 = “stuff” → function returns 2
- s1 = “string”, s2 = “strings” → function returns 6

Solution:

```
int longestMatch(char *s1, char *s2) {
     int i = 0;

     // Keep testing strings until match is not found
     //   or you've reached end of string
     while (i <= strlen(s1)) {

          // Strings don't match--i is one character too many
          if (strncmp(s1, s2, i) != 0)
               return i - 1;

          i++;
     }

     return i - 1;
}
```

6 (cont.)

c. `copyFromPosn()`: Given two strings, `dest` and `source`, as well as an integer `pos`, copy all characters from `source` into `dest`, starting at position `pos` and ending with the end of the source string. Assume `pos` is a valid position within source, and there is enough room in `dest` to hold the source string—you do not need to check for errors. For example:

- source = “string”, pos = 0 → dest = “string”
- source = “string”, pos = 3 → dest = “ing”

Solution:

```
void copyFromPosn(char *dest, char *source, int pos) {
     int i = 0;

     // Keep copying until you reach null at end of source
     while (source[i+pos] != '\0') {
          dest[i] = source[i+pos];
          i++;
     }
}
```

Note: There’s an even shorter version of this function, which takes advantage of the fact that the arguments to the string copy functions are just pointers—just call `strcpy()` with the second argument pointing to the character at position `pos` within `source`:

```
void copyFromPosn(char *dest, char *source, int pos) {
     strcpy(dest, &source[pos]);
}
```