Assignment 3: Paper Pointers

This assignment covers the concept of pointers in C and their relation to simple variables, arrays, and structs. These topics are covered in chapters 11, 12, and section 13.6 in the C book.

Most of the questions are paper and pencil exercises. You can check your answers by using the computer, but try to figure out the results first by drawing pictures. Some ask you to draw pictures. Use a drawing program to create GIF or PNG images, include these images files in the directory with your other files. Do not use any other image format. All you need is boxes, lines, and text. Every MS Windows machine has paint, GNU/Linux systems have Dia, xpaint, kpaint, and others. An alternative is to draw on paper then scan the image to GIF or PNG. Don’t use high resolution. Your non-visual answers must be in plain text files. Do not submit Word files. Do not submit pdf files. Just submit plain text files and GIF or PNG for images. Nothing else.

For all problems, working from simple sketches can help you connect the combinations of *’s, &’s, and brackets to the underlying structure. When in doubt, write short programs to test your ideas.

0. Read the document about indexing and pointing that is posted on the web page for this assignment.

1. Fill in the blank in the following paragraph:

   An address is a number. An address is the position in memory of a piece of storage. A pointer variable is a piece of storage that can store

2. Using the following variable definitions:

   ```c
   char t[3] = "hi";
   ...
   ```

   ```c
   main()
   { ...
   ```

   ```c
   int x, y[4] = { 71, 181, 30, 131 };
   char a[30] = "pointers store addresses.";
   int *q;
   char *p, *s;
   int **r;
   p = a + 4;
   q = &y[0];
   s = &x;
   *s = '0';
   x = 44;
   r = &q;
   *q = q[2] + --x;
   /* write code here to print out values and locations of all vars */
   ```

   Type this program into the computer and add code that prints out the value and location of all eight variables. Then draw a diagram using boxes and arrows that shows the value and location of each variable. For pointer variables, use an arrow to show where the variable points.

3. **Arrays and Pointers**: An array is a chunk of memory that contains a sequence of variables of a single type. For example, the code `int m[5]` creates a sequence of 5 integer variables called m[0], m[1], m[2], m[3], and m[4]. Each of those variables has an address. But what about the array itself? Does an array have an address? It turns out that in C, the name of an array is the address of the first element. The name of an array is an address, not a pointer variable.

   Type in, compile, and run this program:

   ```c
   main()
   { ...
   ```

   ```c
   int m[5] = { 4, 3, 2, 1, 0 };
   int *p;
   p = m + 2;
   printf("m is %lu, &m is %lu, &m[0] is %lu, m[0] is %lu\n",
       m, (unsigned long)&m, (unsigned long)&m[0], m[0]);
   printf("p is %lu, &p is %lu, &p[0] is %lu, p[0] is %lu\n",
       p, (unsigned long)&p, (unsigned long)&p[0], p[0]);
   ```

   Copyright (c) 2017 Bruce Molay

   Revised: Oct 1, 2017 13:53
a) What is the output of the program?
b) Draw a diagram showing m and p and their addresses and values.
c) Explain why the first three values in the first printf are the same.
d) Explain why the first three values in the second printf are not the same.
e) What are the values of \( p[-1] \) and \( p[-2] \)? Explain.

4. Two-Dimensional Arrays, Addresses, and Pointers: The name of an array is the address of the first element. But what about 2D arrays? In class, we said a 2D array is an array of arrays. What are the values of each element in a 2D array? To understand this question, type in, compile, and run this program:

```
main()
{
    char n[4][10] = { "ann", "bob", "carol", "dave" };
    char *p1, *p2;
    p1 = n[2];
p2 = n + 2;

    printf("n is %lu, n[0] is %lu, n[0][0] is %lu
",
            (unsigned long)n, (unsigned long)n[0], (unsigned long)n[0][0]);
    printf("(n+1) is %lu, (n+1)[0] is %lu, (n+1)[0][0] is %lu
",
            (unsigned long)(n+1), (unsigned long)(n+1)[0],
            (unsigned long)(n+1)[0][0]);
    printf("p1 is %lu, p1[0] is %lu
",
            (unsigned long)p1, (unsigned long)p1[0]);
    printf("p1[0][0] is %lu
",
            (unsigned long)p1[0][0]);
    printf("p2 is %lu, p2[0] is %lu
",
            (unsigned long)p2, (unsigned long)p2[0]);
    printf("p2[0][0] is %lu
",
            (unsigned long)p2[0][0]);
}
```
9. *String literals, addresses, and pointers:* The name of an array is not a variable; it is the address of the first element. For example, in `char m[] = "October";`, `m` is a number: the address of this string, but `m` is not a variable: you cannot say `m = "November";`. Of course, the elements in `m` are variables: you can say `m[0] = 'N';`. But what about a string literal like "Hello"? What is the value of "Hello"?

Type in, compile, and run this code:
```c
main()
{
    char *p;
    printf("The string "Hello" has the value %lu\n", (unsigned long)"Hello");
    p = "Hello";
    printf("The variable p has the value %lu\n", (unsigned long)p);
    printf("The value of "Hello"+1 is %lu\n", (unsigned long)("Hello"+1));
    p = "Hello" + 1;
    printf("The variable p now has the value %lu\n", (unsigned long)p);
    printf("The length of p+1 is %d\n", strlen(p+1));
}
```
a) What is the output of this program?
b) Draw a diagram of memory showing "Hello" and `p`
c) Explain the values printed out by the program.

10. a) What is the output of this line of code: `putchar("abcde"[3])`?
b) Why is the expression legal? What does it mean?

11. Pointers are often used to process strings of characters. The sample code and the diagrams given out in class 5 show examples of code that does this and also shows pictures of how the pointers work. Refer to that and use the examples to write two versions of a function that reverses a string of characters and returns the address of the reversed string: first treating the arguments as arrays (i.e. use indexing), then as pointers. The prototype for this function is:
```
char *strrev(char *s)
```
The `strrev()` function replaces the string at `s` with the same string in reverse order and returns the address of the string. The replacement is done in the same array as the original string of characters. No new array is needed or used.

12. Using these definitions and assignments:
```
struct stop {
    char *station;
    int hh, mm;
};
```
```
s1.station = a;
s2.station = b;
p1->station = c;
p1->hh = 8;
s1.mm = 40;
s2.hh = *ip1;
p2->mm = s1.mm;
```
Draw a diagram of the state of all the variables after all the assignments are executed. Use arrows to show where pointers point, and write in values of the other variables.

You do not have to draw diagrams of the variables after each statement is executed. Just draw one diagram showing the variables after all statements are executed.
13. Consider these two array definitions:
   struct stop a[10];  
   struct stop *b[10];

   Explain, using (a) words and (b) diagrams the difference between these arrays.