Using pointers with functions

Recall that our basic use of functions so fare provides for several possibilities. A function can

1. take one or more individual variables as inputs and return a single variable as the output.
2. take no inputs (void) and a single variable as an output.
3. take one or more individual variables as inputs and not return anything as an output (void).
4. take no inputs (void) and not return anything as any output (void)

What's missing in this initial list of things that functions can do is being able to take arrays as inputs and being able to return more than one value as an output. (This would include returning arrays as outputs.)

To get some sense of the problem, let's try to make a function that would swap two integers in memory. We have needed this in several instances and it might be nice to have a re-usable function that does it.

```
//EE 285 - swap (bad)
#include <stdio.h>
void swap( int, int );
int main( void ){
   int x = 3, y = 8;
   printf( "Before swapping: x = %d and y = %d.\n\n", x, y );
   swap( x, y );
   printf( "After swapping:  x = %d and y = %d.\n\n", x, y );
   return 0;
}

void swap( int a, int b ){
   int temp;
   temp = a;
   a = b;
   b = temp;
}
```

```
Before swapping: x = 3 and y = 8.

After swapping:  x = 3 and y = 8.

Program ended with exit code: 0
```

Urk!! That didn't work.

To debug what went wrong, we can add a couple of printf statements to the function.

```
void swap( int a, int b ){

   int temp;

   printf( "Before swapping: a = %d and b = %d.\n", a, b );

   temp = a;
   a = b;
   b = temp;

   printf( " After swapping:  a = %d and b = %d.\n", a, b );
}
```

```
Before swapping: x = 3 and y = 8.

Before swapping: a = 3 and b = 8.
After swapping:  a = 8 and b = 3.

After swapping:  x = 3 and y = 8.

Program ended with exit code: 0
```

So we see that the swapping function *is* working, but the results are not getting back to main. Actually, this should not surprise us at all, once we consider what is going on in memory.

x	3
y	8

x	3
y	8
a	3
b	8
temp	

x	3
y	8
a	3
b	8
temp	3

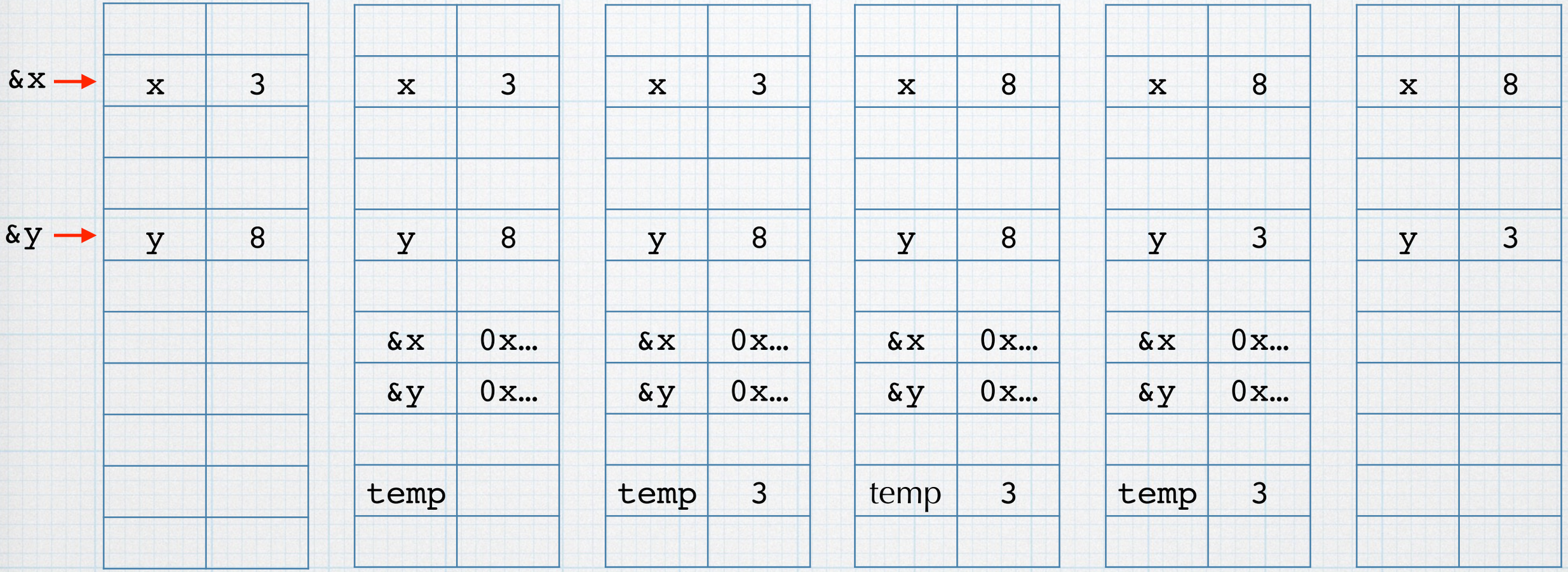
x	8
y	8
a	8
b	8
temp	3

x	8
y	3
a	8
b	3
temp	3

x	8
y	3

Oops! The swapping was localized to the function. Nothing happened back in the main. This was a waste of time.

A light bulb goes off! In the function, try using pointers to the original variables.



Now it would work!

Of course, the code must be modified to pass pointers instead.

```c
//EE 285 - swap (good)
#include <stdio.h>
void swap( int*, int* );
int main( void ){
   int x = 3, y = 8;
   printf( "Before swapping: x = %d and y = %d.\n\n", x, y );
   swap( &x, &y );
   printf( "After swapping:  x = %d and y = %d.\n\n", x, y );
   return 0;
}
```

```c
void swap( int *aPtr, int *bPtr ){
   int temp;
   temp = *aPtr;
   *aPtr = *bPtr;
   *bPtr = temp;
}
```

```
Before swapping: x = 3 and y = 8.

After swapping:  x = 8 and y = 3.

Program ended with exit code: 0
```

Yay!

The realization that we can pass pointers to (and from) functions greatly expands our capabilities. Using pointers, we can "receive" more than one variable from the function. Even better, we can pass (and receive) pointers to arrays. Now we can really start doing some things.

Pass a pointer to an array.

```
//EE 285 - max of an array

#include <stdio.h>

int max( int*, int );

int main( void ){

   int anArray[] =  { 4, 8, 3, 9, 2, 1, 10, 6, 5, 72, 0 };
   int maxValue;

   maxValue = max( anArray, 11 );

   printf( "The maximum value is %d.\n\n", maxValue );

   return 0;
}

int max( int a[], int size ){

   int i, biggie = 0;

   for(i = 0; i < size; i++ )
      if( a[i] > biggie)
         biggie = a[i];

   return biggie;
}
```

```
The maximum value is 72.

Program ended with exit code: 0
```

```c
//EE 285 - average of an array

#include <stdio.h>

double average( int*, int );

int main( void ){

   int anArray[] =  { 4, 8, 3, 9, 2, 1, 10, 6, 5, 12, 0 };
   double avgValue;

   avgValue = average( anArray, 11 );

   printf( "The average value is %5.3lf.\n\n", avgValue );

   return 0;
}

double average( int* aPtr, int size ){

   int i;
   double sum = 0;

   for(i = 0; i < size; i++ )
         sum = sum + *(aPtr + i);

   return sum/size;
}
```

```
The average value is 5.455.

Program ended with exit code: 0
```

Use functions to bubble sort an array. (Bubble sort function is on following page.)

```c
//EE 285 - bubble sort an array

#include <stdio.h>

void bubbleSort( int*, int );
void printArray( int*, int );

int main( void ){

   int anArray[] =  { 4, 8, 3, 9, 2, 1, 10, 6, 5, 12, 0 };

   printArray( anArray, 11 );
   bubbleSort( anArray, 11 );
   printArray( anArray, 11 );

   return 0;
}

/*  printArray function  */

void printArray( int a[], int size ){

   int i;

   printf( "The array is { " );
   for(i = 0; i < size ; i++ )
      printf( "%d ", a[i] );

   printf( "}\n\n" );
}
```

```c
/*  bubblesort function  */

void bubbleSort( int a[], int size ){

   int i, j, swap;

   for(i = 0; i < size - 1 ; i++ ){
      for( j = 0; j < size - 1; j++){
         if( a[j+1] > a[j] ){
            swap = a[j];
            a[j] = a[j+1];
            a[j+1] = swap;
         }
      }
   }
}
```

```
The array is { 4 8 3 9 2 1 10 6 5 12 0 }

The array is { 12 10 9 8 6 5 4 3 2 1 0 }

Program ended with exit code: 0
```

Write a function that generates a string with random characters. (main() is below and the functions are on the next page.)

```
//EE 285 - generate a random array

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void randCharArray( char*, int );
void printCharArray( char* );

int main( void ){

   const int SIZE = 11;     //10 characters + '\0'
   char cArray[ SIZE ];

   srand( (int)time(0) );

   randCharArray( cArray, SIZE );
   printCharArray( cArray );

   return 0;
}
```

Note: Printable ASCII characters have decimal values from 32 to 126. (ASCII code 32 coresponds to space. See https://en.wikipedia.org/wiki/ASCII .)

```c
/*  print a character array  */

void printCharArray( char a[] ){
   int i = 0;
   printf( "The array is: " );
   while( a[i] != '\0' ){
      printf( "%c ", a[i] );
      i++;
   }
   printf( "\n\n" );
}

/*  generate random character array  */

void randCharArray( char a[], int s ){
   int i;
   for(i = 0; i < s - 1 ; i++ )
      a[i] = rand()%94 + 32;

   a[i] = '\0';     //Add the trailing 0.
}
```

```
The array is: 7 U # Q n < y 3 z |

Program ended with exit code: 0
```

`Sizeof` function

Use this to get the size (number of bytes) of something stored in memory. With this, we do not need to know ahead of time how big an array is. Below is the bubble sort `main()` that uses `sizeof()` to figure out how big (how many elements) are in the array. The functions are unchanged and not included below.

```
//EE 285 - bubble sort an array
#include <stdio.h>

void bubbleSort( int*, int );
void printArray( int*, int );

int main( void ){

   int anArray[] =  { 4, 8, 3, 9, 2, 1, 10, 6, 5, 12, 0 };
   int length;

   length = sizeof(anArray) / sizeof(anArray[0]);

   printf( "The array size is %d.\n\n", length );

   printArray( anArray, length );
   bubbleSort( anArray, length );
   printArray( anArray, length );

   return 0;
}
```

```
The array size is 11.

The array is { 4 8 3 9 2 1 10 6 5 12 0 }

The array is { 12 10 9 8 6 5 4 3 2 1 0 }

Program ended with exit code: 0
```