To:
Tenouk

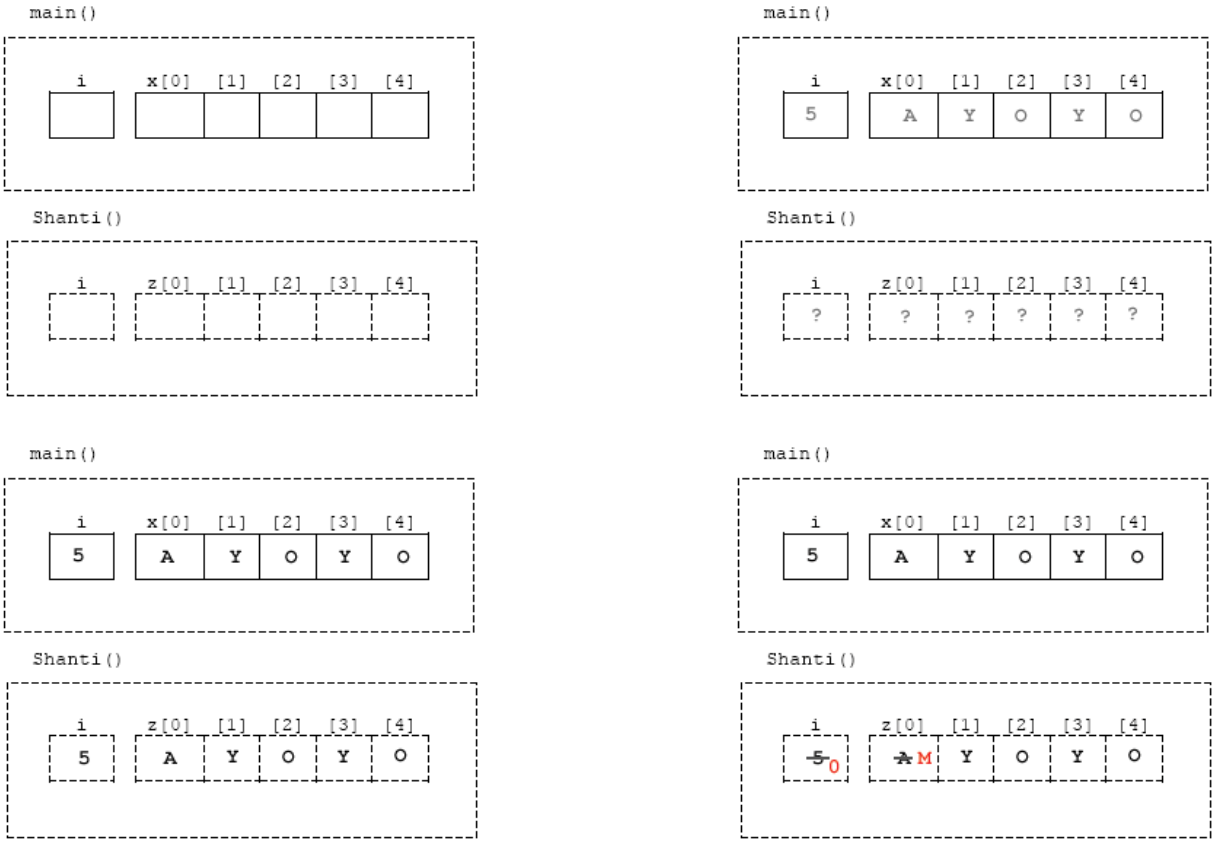
C LAB WORKSHEET 12a
C & C++ Functions Part 2 (with no return values)

1. More on C/C++ User-defined Functions.
2. Functions, loops and arrays.
3. Functions and character strings.
4. Passing arguments by values and references (a pointer to memory addresses).
5. Tutorial references that should be used together with this worksheet are starting from C/C++ function part 1, function part 2, function part 3 and function part 4 and C/C++ main().

10. The following diagram is misleading, though it seems correct. There are no loops this time. We are observing how functions handle array and scalars (non-arrays). Show the output and answer the questions.

```
#include <stdio.h>

// function prototype
void Shanti(int, char[ ]);

void main(void)
{
  int i = 5;
  char x[ ] = "AYOYO";
  printf("i = %d, x = %s\n", i, x);   // Statement 1
  // calling Shanti, notice the array sent, just the array name...
  Shanti(i, x);                       // Statement 2
  printf("i = %d, x = %s\n", i, x);
}

// function definition
void Shanti(int i, char z[ ])
{
  printf("\nIn Shanti, i = %d, z = %s\n", i, z);
  i = 0;
  z[0] = 'M';                                   // Statement 3
  printf("In Shanti, i = %d, z = %s\n", i, z);  // Statement 4
}
```

```
i = 5, x = AYOYO

In Shanti, i = 5, z = AYOYO
In Shanti, i = 0, z = MYOYO
i = 5, x = MYOYO
Press any key to continue . . .
```



a. The outlined box for main() shows two variables as they appear in memory, i and a string called x [ ]. Show the contents of each one before main() calls Shanti() in Statement 1. Remember that a string is a set of characters with a null character at the end. **Ans:** Refer top-right diagram.
b. The outlined box for Shanti() shows its two arguments. This representation is incorrect and in next experiment we will clarify it. Show the contents of the variables in Shanti() when it is first called

as in Statement 2. **Ans:** Refer bottom-left diagram.

c. In the outlined box for Shanti(), cross off the items that are changed and under them show the new values, as in Statement 3. One should be 0 and the other an 'M'. Is this change confirmed by the last printf() in Shanti()? **Ans:** Yes. Refer bottom-right diagram.

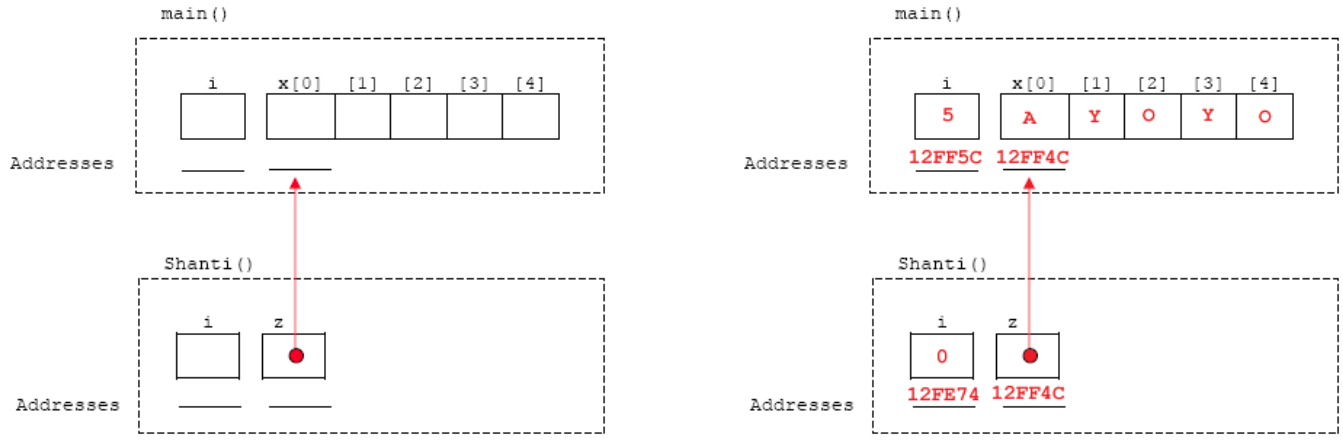
d. In previous experiments we had already confirmed that i in main() is a different variable than the i in the other function. According to our incorrect diagram, when control comes back to main(), for the last printf(), as in Statement 4, the value of i should still be 5 and not the 0 that was assigned in Shanti(). Does the printf() confirm that? **Ans:** Yes the i still 5, seen through the program output.

e. According to our incorrect diagram, x[0] should still have a value of 'A' and not 'M' that was assigned in Shanti(). Does the last printf() in main() confirm that? Your answer here should be no! We will correct the diagram in the next experiment. **Ans:** No. For an array passed to a function, the array changes in the function will take effect to the actual array data although the function body was exited.

11. Now, the following diagram is correct. We are printing out addresses. If two variables are stored in the same memory address, then it is the same location. Otherwise, they are stored in different locations. Show the output and answer the questions.

```
#include <stdio.h>

// function prototype
void Shanti(int, char[ ]);

void main(void)
{
  int i = 5;
  char x[ ] = "AYOYO";
  printf("i = %d, x = %s, &i = %p, &x[0] = %p\n", i, x, &i, &x[0]);
  // calling Shanti, notice the array, just the array name...
  Shanti(i, x);
  printf("i = %d, x = %s, &i = %p, &x[0] = %p\n", i, x, &i, &x[0]);
}

// function definition
void Shanti(int i, char z[ ])
{
  printf("\nIn Shanti(), i = %d, z = %s, &i = %p, &z[0] = %p\n", i, z, &i, &z[0]);
  i = 0;
  z[0] = 'M';

  printf("In Shanti(), i = %d, z = %s, &i = %p, &z[0] = %p\n\n", i, z, &i, &z[0]);
}
```

```
i = 5, x = AYOYO, &i = 0012FF5C, &x[0] = 0012FF4C

In Shanti(), i = 5, z = AYOYO, &i = 0012FE74, &z[0] = 0012FF4C
In Shanti(), i = 0, z = MYOYO, &i = 0012FE74, &z[0] = 0012FF4C

i = 5, x = MYOYO, &i = 0012FF5C, &x[0] = 0012FF4C
Press any key to continue . . .
```



a. In the diagram, inside the outlined box for main(), Write the value of i in its box and its address under it. Write the value of x[0] through x[4] in their boxes. Write the address of x[0] under its box. (Different computers may have different memory addresses.) **Ans:** Refer to the diagram on the right.

b. In the diagram, inside the outlined box for Shanti(), write the value of i in its box and its address under it. Write the address of z[0] under it. **Ans:** Refer to diagram on the right.

c. Are the addresses of both i's different? If they are, then they are stored in separate locations in memory, just as shown in the diagram. **Ans:** Yes, they are different.

d. Are the addresses of x[ ] and z[ ] the same? If so, then they are really the same location, just as shown in the diagram. If z[0] is assigned 'M' in Shanti(), then it changes x[0] in main() because they are the same location. **Ans:** Yes, both addresses are same, &x[ ] and &z[ ] are same.

e. Why doesn't changing i in Shanti() change i in main()? **Ans:** Because both i stored at different memory locations.

f. Is the i in main() stored in the same memory address as the i stored in Shanti()? Your conclusion here should be no! **Ans:** Of course no based on the program output.

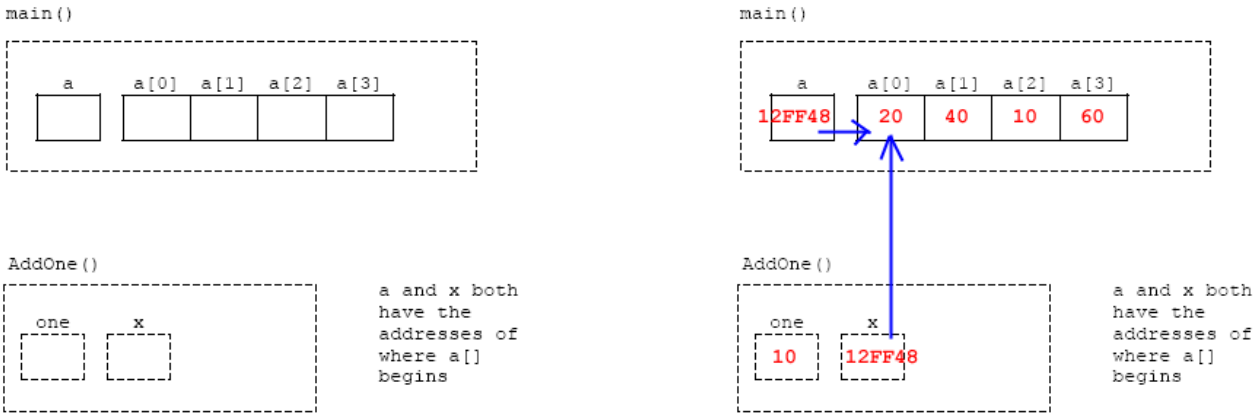
g. Is x[ ] in main() stored in the same memory address as is the array z[ ] stored in Shanti()? Your conclusion here should be yes. z[0] is the same location in memory as x[0]. If you change one, you change the other. **Ans:** Yes, both store their data at the same location. Here we can see that **for an array**, **only a reference of memory address has been passed to the function**, not the actual value.

12. When passing an array to a function, the changes the function may make will affect the first array because the function has the address of that array. The following is another experiment illustrating that concept. We will call single-element variables such as i (int, float,

double, char etc), **scalars**, to contrast them with arrays. After running the experiment, show the output and fill the boxes in the diagram, answer the questions.

```
#include <stdio.h>

// function prototype
void AddOne(int, int[ ]);

void main(void)
{
  // local to main()...
  int i, a[4] = {20, 40, 10, 60};
  printf("In main(), a = %p, &a[0] = %p\n", a, &a[0]);
  // call function AddOne()...
  AddOne(a[2], a);
  for(i = 0; i <= 3; i = i + 1)
    printf("a[%d] = %d\n", i, a[i]);
}

// function definition
void AddOne(int one, int x[ ])
{
  // local to AddOne()...
  int i;
  printf("In AddOne(), x = %p, &x[0] = %p\n", x, &x[0]);
  for(i = 0; i <= 3; i = i + 1)
      x[i] = x[i] + one;
}
```

```
In main(), a = 0012FF48, &a[0] = 0012FF48
In AddOne(), x = 0012FF48, &x[0] = 0012FF48
a[0] = 30
a[1] = 50
a[2] = 20
a[3] = 70
Press any key to continue . . .
```



a. Was a scalar or an array passed when main() passed a[2] to AddOne()? **Ans:** A scalar. a[2] refer to the third array element and in this case it is 10.
b. Was a scalar or an array passed when main() passed a to AddOne()? **Ans:** An array. An array name is a pointer to the first array's element.
c. What was the array called in main()? **Ans:** a
d. What was the array called in AddOne()? **Ans:** x
e. When AddOne() changed the array x[ ] by adding 1 to it, did the array a[ ] change in main()? Why? **Ans:** Yes. This is because those different arrays, a[ ] and x[ ] are referring to the same array.
f. Are the addresses stored in a and x different or the same? **Ans:** Of course they are same.

More Questions and Activities on Functions

1. Show the output for the following programs.

   a. State what the following program tries to demonstrate.

```
#include <stdio.h>

void Questions(void);

void main(void)
{
    int i;
    int j = 1;
    for(i = 3; i <= 5; ++i)
    {
        printf("Call #%d - ", j++);
        // function call in the loop...
        Questions();
    }
}

void Questions(void)
{
    printf("Did you understand what the function is?\n");
}
```

```
Call #1 - Did you understand what the function is?
Call #2 - Did you understand what the function is?
Call #3 - Did you understand what the function is?
Press any key to continue . . .
```

A function calls is enclosed in the for loop. The number of calls depend on the number of loop iteration and in this case it is 3 times.

b. What this program tries to show?

```
#include <stdio.h>

void Questions(void);

void main(void)
{
    // first call
    Questions();
    // second call
    Questions();
}

void Questions(void)
{
    int j;
    // three iterations...
    for(j = 1; j <= 3; ++j)
        printf("Pass #%d - What function?\n", j);
    printf("\n");
}
```

```
Pass #1 - What function?
Pass #2 - What function?
Pass #3 - What function?

Pass #1 - What function?
Pass #2 - What function?
Pass #3 - What function?

Press any key to continue . . .
```

There are two function calls from main() and each call will have three iterations of the for loop.

c. What this program tries to show?

```
#include <stdio.h>

void Questions(void);

void main(void)
{
    // local to main()...
    int i, p = 1;
    for(i = 3; i <= 5; ++i)
    {
        printf("Call #%d\n", p++);
        Questions();
    }
}

void Questions(void)
{
    // local to Question()...
    int j;
    for(j= 1; j <= 2; ++j)
        printf("Pass #%d - What function?\n", j);
    printf("\n");
}
```

```
Call #1
Pass #1 - What function?
Pass #2 - What function?

Call #2
Pass #1 - What function?
Pass #2 - What function?

Call #3
Pass #1 - What function?
Pass #2 - What function?

Press any key to continue . . .
```

There are three function calls from main() depicted by the for loop iterations and each call will have two iterations in the function body.

d. You can use use 4, 1 and 3 as the sample input data for the following program. What this program tries to show?

```
#include <stdio.h>

void Questions(void);

void main(void)
{
    int i;
    // loop for three times...
    for(i = 3; i <= 5; ++i)
        Questions();
}

void Questions(void)
{
    int j;
    // prompt for the iteration number...
    printf("Enter an integer: ");
    scanf_s("%d", &j);
    // do the iteration...
    for( ; j != 0; --j)
        printf("What function?\n");
    printf("\n");
}
```

```
Enter an integer: 3
What function?
What function?
What function?

Enter an integer: 2
What function?
What function?

Enter an integer: 1
What function?

Press any key to continue .
```

There are three function calls from main() and each call will have the number of iterations based on the user inputs.

e. Enter 4 as a simple input data. What this program tries to show?

```
#include <stdio.h>

void Busted(int);

void main(void)
{
    int j;
    printf("Enter an integer: ");
    scanf_s("%d", &j, 1);
    for( ; j != 0; --j)
        // calling Busted() in loop
        Busted(j);
    printf("\n");
}

void Busted(int k)
{
    printf("k = %d\t", k);
}
```

```
Enter an integer: 4
k = 4   k = 3   k = 2   k = 1
Press any key to continue . . .
```

A function call in for loop and every call will pass an integer value that got printed in the callee function body. The for loop in the main body iterates descendingly.

f. Assume the user enters a 4 as a simple input data. What this program tries to show?

```
#include <stdio.h>

void Rusted(char[ ]);

void main(void)
{
    // all work done in function Rusted()...
    Rusted("Test Test");
    printf("\n");
}

void Rusted(char x[ ])
{
    int j;
    printf("Enter an integer: ");
    scanf_s("%d", &j);
    for(; j != 0; --j)
        printf("In Rusted(), x = %s\n", x);
}
```

```
Enter an integer: 4
In Rusted(), x = Test Test
In Rusted(), x = Test Test
In Rusted(), x = Test Test
In Rusted(), x = Test Test

Press any key to continue . .
```

A function call from main() that passes a character string and callee will print the number of character string based on the user input.

g. What this program tries to show?

```
#include <stdio.h>

void Mustard(int, int);

void main(void)
{
    int i = 50, a[3] = {90, 80, 60};
    Mustard(i, a[0]);
    printf("In main(), i = %d, a[0] = %d\n", i, a[0]);
}

void Mustard(int q, int w)
{
    printf("In Mustard(), q = %d, w = %d\n", q, w);
    q = 0;
    w = 0;
}
```

```
In Mustard(), q = 50, w = 90
In main(), i = 50, a[0] = 90
Press any key to continue . . .
```

A function call from main() passes two integers. These integers got printed in the called function and then the values are re-assigned. However we can see that those variables are local to their functions respectively.

h. What this program tries to show? You may draw a diagram to help you.

```
#include <stdio.h>

void Mustard(int, int [ ]);

void main(void)
{
    int q = 50, a[3] = {90, 80, 60};
    Mustard(q, a);
    printf("In main(), q = %d, a[0] = %d\n", q, a[0]);
}

void Mustard(int q, int w[ ])
{
    printf("In Mustard(), q = %d, w[0] = %d\n", q, w[0]);
    q = 0;
    w[0] = 0;
}
```

```
In Mustard(), q = 50, w[0] = 90
In main(), q = 50, a[0] = 0
Press any key to continue . . .
```

Contradict to the previous example, an array passed to a function, if there are changes, they are reflected to the original or actual array data that was passed. This is not in case of scalar variables such as int.

i. What this program tries to show?

```
#include <stdio.h>

void DropIn(char [ ], char [ ], int, int, int);

void main(void)
{
    char a[30] = "Jack The Ripper";
    DropIn(a, "Jumping ", 5, 10, 8);
    printf("a = %s\n", a);
}

void DropIn(char str[ ], char instr[ ], int start, int len, int ilen)
{
    int i, j;
    str[len + ilen] = '\0';
    // the two statements before the first semicolon initialize the loop
    // and the two statements after the second semicolon
    // are executed for each iteration. --i similar to i = i - 1
    for(i = len -1, j = len + ilen -1; len - ilen - 1 != i; --i, --j)
        str[j] = str[i];
    for(i = 0, j = start; ilen != i; ++i, ++j)
        str[j] = instr[i];
}
```

```
a = Jack Jumping The R
Press any key to continue . .
```

A function calls from main() passes an array, a character string and integers. These arguments then manipulated in the callee, returning a result to the main().

2. For each of the following questions, write a complete code listing that includes the main() as well as the other function required to test each problem.

a. Have main() read in a string and an integer called num. Call a function num times, passing the string with it. The function should print the string once each time it is called.

```
#include <stdio.h>

void PrintMe(char [ ]);

void main(void)
{
   char mystring[10];
   int i, num;

   printf("Enter a count: ");
   // scanf("%d", &num);
   scanf_s("%d", &num, 1);
   printf("Enter a string: ");
   scanf_s("%s", &mystring, sizeof(mystring));
   for(i=1;i <=num;i++)
      PrintMe(mystring);
   printf("\n");
}

void PrintMe(char hold_string[ ])
{
   printf("I'm printing a \"%s\"\n", hold_string);
}
```

```
Enter a count: 5
Enter a string: Testing
I'm printing a "Testing"
I'm printing a "Testing"
I'm printing a "Testing"
I'm printing a "Testing"
I'm printing a "Testing"

Press any key to continue . .
```

b. Follow the instructions for previous program, but have main() pass both the string and the num to the function only once. Let the function set up the loop that will print that string num number of times.

```
#include <stdio.h>

void PrintMe(int num, char [ ]);

void main(void)
{
   char mystring[10];
   int num;

   printf("Enter a count: ");
   // scanf("%d", &num);
   scanf_s("%d", &num, 1);
   printf("Enter a string: ");
   // scanf("%s", &mystring);
   scanf_s("%s", &mystring, sizeof(mystring));
   PrintMe(num, mystring);
   printf("\n");
}

void PrintMe(int hold_num, char hold_string[ ])
{
   int i;
   for(i=1; i<=hold_num;i++)
      printf("I'm printing a \"%s\"\n", hold_string);
}
```

```
Enter a count: 5
Enter a string: Testing2
I'm printing a "Testing2"
I'm printing a "Testing2"
I'm printing a "Testing2"
I'm printing a "Testing2"
I'm printing a "Testing2"

Press any key to continue . .
```

c. The function should receive three integers and print their average. main () should read in three integers and call the function by passing these integers.

```
#include <stdio.h>

void MyAverage(int num1, int num2, int num3);

void main(void)
{
   int num1, num2, num3;

   printf("Enter three integers: ");
   // scanf("%d%d%d", &num1, &num2, &num3);
   scanf_s("%d%d%d", &num1, &num2, &num3, 1, 1, 1);
   MyAverage(num1, num2, num3);
}

void MyAverage(int hold_num1, int hold_num2, int hold_num3)
{
   float avg;

   avg = (float)(hold_num1 + hold_num2 + hold_num3)/3;
   printf("The average of three numbers is: %.2f\n", avg);
}
```

```
Enter three integers: 20 10 40
The average of three numbers is: 23.33
Press any key to continue . . .
```

d. The function should receive an array of floats and a scalar float (call it "scalar"). It should then print all the elements in the array greater than "scalar".

```
#include <stdio.h>

void GraterThan(float, float[ ]);

void main(void)
{
   float scalar = 2.50, num2[5] = {7.45f, 2.19f, 6.45f, 4.32f, 5.10f};
   GraterThan(scalar, num2);
}

void GraterThan(float hold_scalar, float hold_num2[5])
{
   int i;
   for(i=0;i<=4;i++)
   if(hold_num2[i] > hold_scalar)
      printf("The greater than %.2f are %.2f\n", hold_scalar, hold_num2[i]);
}
```

```
The greater than 2.50 are 7.45
The greater than 2.50 are 6.45
The greater than 2.50 are 4.32
The greater than 2.50 are 5.10
Press any key to continue . . .
```

e. The function should receive an array of floats and a scalar float called "scalar". The function should add the scalar to each element of the array. Have main() print the array.

```
#include <stdio.h>

void AddScalar(float, float[ ]);

void main(void)
{
   float scalar = 2.50, num2[5] = {7.45f, 2.19f, 6.45f, 4.32f, 5.10f};
   AddScalar(scalar, num2);
}

void AddScalar(float hold_scalar, float hold_num2[5])
{
   int i;
   for(i=0;i<=4;i++)
      printf("The new array are %.2f\n", hold_scalar + hold_num2[i]);
}
```

```
The new array are 9.95
The new array are 4.69
The new array are 8.95
The new array are 6.82
The new array are 7.60
Press any key to continue . . .
```

f. The function should receive a character string. After counting the characters in the string until it finds the null character, it should print that count. For example, if main() passed “starlight” to the function, it should print its length, 9.

```
#include <stdio.h>

void Counting(char[ ]);

void main(void)
{
   // 49 + '\0' storage
   char mystring[50];

   printf("Enter a string: ");
   // for string containing whitespaces, we need to
   // use other functions such as gets()/gets_s()
   // scanf("%s", &mystring);
   scanf_s("%s", &mystring, sizeof(mystring));
   Counting(mystring);
}

void Counting(char hold_mystring[ ])
{
   int i;

   for(i=0; hold_mystring[i] != '\0'; i++);
      printf("Length of %s is %d\n", hold_mystring, i);
}
```

```
Enter a string: A_string_sample
Length of A_string_sample is 15
Press any key to continue . . .
```

We can use a built-in function, strlen() to accomplish this task as shown below.

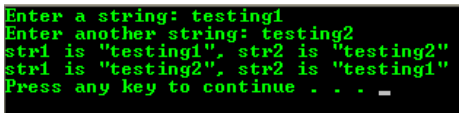
```
#include <stdio.h>
#include <string.h>

void Counting(char[ ]);

void main(void)
{
   // 49 + '\0' storage
   char mystring[50];

   printf("Enter a string: ");
   // for string containing whitespaces, we need to
   // use other functions such as gets()/gets_s()
   // scanf("%s", &mystring);
   scanf_s("%s", &mystring, sizeof(mystring));
   Counting(mystring);
}

void Counting(char hold_mystring[ ])
{
   // secure version is strnlen(string, buffer_size)
   printf("The number of characters are %d\n", strlen(hold_mystring));
}
```

```
Enter a string: null_terminated_strings
The number of characters are 23
Press any key to continue . . .
```

g. The function called Swap() should receive two strings and exchange them. For example, if main() has initialized two strings called a[ ] and b [ ], then after calling Swap(), the string which was in a[ ] will be in b[ ] and vice versa. Remember that when you pass an array to a function, you are passing its address.

```
#include <stdio.h>

void Swap(char[ ], char[ ]);

void main(void)
{
   char str1[20], str2[20];

   printf("Enter a string: ");
   // scanf("%s", &str1);
   scanf_s("%s", &str1, sizeof(str1));
   printf("Enter another string: ");
   // scanf("%s", &str2);
   scanf_s("%s", &str2, sizeof(str2));
   printf("str1 is \"%s\", str2 is \"%s\"\n", str1, str2);
   // pass str1 and str2
   Swap(str1, str2);
}

void Swap(char hold_str1[20], char hold_str2[20])
{
   int i;
   char hold_temp[20];

   for(i=0;i <= 19;i++)
   {
      // hold str1 temporarily
      hold_temp[i] = hold_str1[i];
```

```
        // copy all str2 into str1, overwriting str1
        // str1 and str2 holding same data, str2
        hold_str1[i] = hold_str2[i];
        // copy all str1 from hold_temp into str2
        hold_str2[i] = hold_temp[i];
    }
    printf("str1 is \"%s\", str2 is \"%s\"\n", hold_str1, hold_str2);
}
```



We can also use strcpy(), wcscpy(), _mbscpy() or secure version, strcpy_s(), wcscpy_s(), _mbscpy_s() to copy a string.

www.tenouk.com
