To:
Tenouk

C LAB WORKSHEET 12a_1
C & C++ Functions Part 3 (with no return values)

1. More on C/C++ User-defined Functions activities, questions and answers.
2. Functions, loops and arrays.
3. Functions and strings.
4. Passing arguments by values and by references.
5. Function calls and program flows.
6. 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().

More Practice

1. Study the following function construct.

```
#include <stdio.h>

// prototype for Testing(), needed by C++ standard
// take note about the semicolon at the end of the statement...
void Testing(void);

// definition for main(). The first void means main() returns nothing
// the second void means main() receive nothing
void main(void)
{
  // main() is calling Testing() and not passing any arguments
  // jump to Testing()...
  Testing();
}

// definition for Testing(), a user defined function
// the second void means Testing() receive nothing
// all work done or completed in the function body
void Testing(void)
{
  // Testing() is calling printf() and passing one argument, a string
  // this is a built in function translated to the defined task through
  // the stdio.h header file...
  printf("Function that just displaying a string!\n");
  // back to main()...
}
```

```
Function that just displaying a string!
Press any key to continue . . .
```

In the program the main() calling a function called Testing(). Testing() in turn, calls printf(). When main() calls Testing(), no arguments or parameters are passed, which is noted by the empty parentheses. When Testing() calls printf(), the argument "Function that just displaying a string!\n" is passed. When writing a function other than main(), you should provide a prototype at the top of the program (or before the function is defined), declaring what the function **receives** and what it **returns**. Think of the prototype as a declaration for a variable, such as , int i; where the type of the variable is specified. The prototype is also declaring the function type, void in this case. For the following statement, number the events in order starting from 1.

_____ - Testing() calls printf().
_____ - We/system call main()
_____ - main() begins to execute by calling Testing()
_____ - End of the program.
_____ - Execution returns to main() from Testing().

Ans:

__3__ - Testing() calls printf().
__1__ - We/system call main()
__2__ - main() begins to execute by calling Testing()
__5__ - End of the program.
__4__ - Execution returns to main() from Testing().

2. Check off (yes or no) the items requiring semicolons at the end of their lines.

_____ - Calling a function.
_____ - Heading for a function definition.
_____ - Prototype for a function.

__Yes__ - Calling a function.
__No__ - Heading for a function definition.
__Yes__ - Prototype for a function.

In the header for a function definitions, which void, the first one or the one in the parentheses, means that the function isn't receiving any arguments? _____
Which one means that the function isn't returning any values? _____

void in parentheses.
The first one.

3. Rewrite only the main() in the previous program example, which calls Testing() two times. Also provide the output.

```
#include <stdio.h>

void Testing(void);

void main(void)
{
   int i;
   for(i=1;i <=2;i++)
      Testing();
}

void Testing(void)
{
   printf("Function that just displaying a string!\n");
```

```
}
```

```
Function that just displaying a string!
Function that just displaying a string!
Press any key to continue . . .
```

4. Next, rewrite only Testing() in question 1, so that the output will be the same as shown in the previous question (question 3).

```
#include <stdio.h>

void Testing(void);

void main(void)
{
   Testing();
}

void Testing(void)
{
   int i;
   for(i=1;i <=2;i++)
      printf("Function that just displaying a string!\n");
}
```

```
Function that just displaying a string!
Function that just displaying a string!
Press any key to continue . . .
```

5. Rewrite only the main() in question 1 so that Testing() is called in a loop four times. Use ++i or i++, which is the abbreviated version of i = i + 1. The output will be as shown below.

```
--------Output---------
nice!
nice!
nice!
nice!
```

```
#include <stdio.h>

void Testing(void);

void main(void)
{
   int i;
   for(i=1;i <=4;++i)
      Testing();
}

void Testing(void)
{
   printf("nice!\n");
}
```

```
nice!
nice!
nice!
nice!
Press any key to continue . .
```

6. Rewrite only Testing() in question 1 so that printf() is called in a loop four times. The output will be the same as in previous question (question 5).

```
#include <stdio.h>

void Testing(void);

void main(void)
{
   Testing();
}

void Testing(void)
{
   int i;
   for(i=1;i <=4;++i)
      printf("nice!\n");
}
```

```
nice!
nice!
nice!
nice!
Press any key to continue . .
```

7. In your solution 5, Testing() is called how many times? Once we are in Testing(), printf() is called how many times? In solution 6, Testing() is called how many times? printf() is called how many times? What is the total number of times that printf() is called in each solution?

In solution 5, Testing() was called 4 times and while in Testing(), printf() was called once. Total number of times that printf() was called in each solution is 4. In solution 6, Testing was called once and while in Testing(), printf() was called 4 times.

8. Now we want Testing() to accept any string as an argument so that, depending on what string we pass to Testing(), that string will be printed in a loop. We wouldn't be restricted to printing only "nice!". I have given you the prototype (notice that the name of the argument is optional) and the definition for Testing(). str[ ] is the argument that will be printed four times in a loop. Can you write main() so that when Testing() is called, you pass the string "nice!" to it? Also show the output. Notice the brackets have no numbers in them.

```
#include <stdio.h>

void Testing(char []);

...
...
...

void Testing(char str[])
{
   int i;
   for(i = 1; i <= 4; ++i)
```

```
#include <stdio.h>

void Testing(char [ ]);

void main(void)
{
   Testing("nice!");
}

void Testing(char str[ ])
{
   int i;
   for(i = 1; i <= 4; ++i)
      printf("str = %s ", str);
   printf("\n");
}
```

```
str = nice!  str = nice!  str = nice!  str = nice!
Press any key to continue . . .
```

```
        printf("str = %s  ", str);
    printf("\n");
}
```

9. Now rewrite main() so that it calls Testing() twice, once as it passes “nice!” and the next time as it passes “sweet!” as arguments. Show the output again.

```
#include <stdio.h>

void Testing(char [ ]);

void main(void)
{
   Testing("nice!");
   Testing("sweet!");
}

void Testing(char str[ ])
{
   int i;
   for(i = 1; i <= 4; ++i)
      printf("str = %s ", str);
   printf("\n");
}
```

```
str = nice!  str = nice!  str = nice!  str = nice!
str = sweet!  str = sweet!  str = sweet!  str = sweet!
Press any key to continue . . .
```

10. Write main() so that when using scanf()/scanf_s(), it gets a string from the user and passes that string to Testing(). Thus, it can print that string four times. Show the output for the string “kind” being read.

```
#include <stdio.h>

void Testing(char [ ]);

void main(void)
{
   char str[10];

   printf("Enter a string: ");
   // scanf("%s", &str);
   scanf_s("%s", &str, sizeof(str));
   Testing(str);
}

void Testing(char str[ ])
{
   int i;
   for(i = 1; i <= 4; ++i)
      printf("%s ",str);
   printf("\n");
}
```

```
Enter a string: kind
kind kind kind kind
Press any key to continue . . .
```

11. Write the main(), so that using a loop, three strings are read in and Testing() is called each time. Here is a sample output:

```
---------Output---------
Enter a small string: kind
kind  kind  kind  kind
Enter a small string: good
good  good  good  good
Enter a small string: cute
cute  cute  cute  cute
```

```
#include <stdio.h>

void Testing(char []);

void main(void)
{
   char str[10], str1[10], str2[10];

   printf("Enter a small string: ");
   // scanf("%s", &str);
   scanf_s("%s", &str, sizeof(str));
   Testing(str);
   printf("Enter a small string: ");
   // scanf("%s", &str1);
   scanf_s("%s", &str1, sizeof(str1));
   Testing(str1);
   printf("Enter a small string: ");
   // scanf("%s", &str2);
   scanf_s("%s", &str2, sizeof(str2));
   Testing(str2);
}

void Testing(char str[])
{
   int i;
   for(i = 1; i <= 4; ++i)
      printf("%s ",str);
   printf("\n");
}
```

```
Enter a small string: kind
kind kind kind kind
Enter a small string: good
good good good good
Enter a small string: cute
cute cute cute cute
Press any key to continue . .
```

12. Now let us continue to add more flexibility to Testing(). We also want the calling program to determine how many times the string will be printed. The calling program will pass two arguments, a string to be printed and an integer, that determine the number of times that string will be printed. First write the prototype.

```
void Testing(char [ ], int);
```

13. Next, write the definition for Testing(). Call the string str[ ] as before and call the integer num. Remember that there is no semicolon for the function header.

```
void Testing(char str[ ], int num)
{
   int i;
   for(i = 1; i <= num; ++i)
      printf("%s ",str);
   printf("\n");
}
```

14. Write main() to go with our new Testing(). Have main() call Testing() twice. The first time have it pass “forgiving” and 3. The second time have it pass “patience” and 2. Also show the output.

```
void main(void)
{
   Testing(“forgiving”, 3);
   Testing(“patience”, 2);
}
```

The following is a complete code:

```
#include <stdio.h>

void Testing(char [ ], int);

void main(void)
{
   Testing("forgiving", 3);
   Testing("patience", 2);
}

void Testing(char str[ ], int num)
{
   int i;
   for(i = 1; i <= num; ++i)
      printf("%s ",str);
   printf("\n");
}
```

```
forgiving forgiving forgiving
patience patience
Press any key to continue . . .
```

15. Next, let us start working with arrays. main() has two integer arrays called A[3] and B [3]. Have main() send A[i] and B[i] to a function called Largest(). This function should print the larger of the two integers. Write the function, the prototype and the main(). A sample output is given below for the following input samples: A[i] = 20, B[i] = 30; A[i] = 40, B[i] = 50; A[i] = 50, B[i] = 60.

```
------Output-----
The largest is 30
The largest is 50
The largest is 60
```

```
#include <stdio.h>

void Largest(int [ ], int [ ]);

void main(void)
{
   int A[3] = {40, 70, 20}, B[3]={45, 35, 75};

   // pass array's element pointers or addresses
   Largest(&A[0], &B[0]);
   Largest(&A[1], &B[1]);
   Largest(&A[2], &B[2]);
}

void Largest(int hold_arr1[ ], int hold_arr2[ ])
{
   int i =0, largest;

   if(hold_arr1[i] > hold_arr2[i])
      largest = hold_arr1[i];
   else
      largest = hold_arr2[i];
   printf("The largest is %d\n", largest);
}
```

```
The largest is 45
The largest is 70
The largest is 75
Press any key to continue .
```

16. Next, re-write only the main(), Largest() being the same as before. Using loop, have main() send all three pairs of numbers, one pair at a time, to Largest().

We still need to edit some part of the Largest().

```
#include <stdio.h>

void Largest(int [ ], int [ ]);

void main(void)
{
   int A[3] = {40, 70, 20}, B[3]={45, 35, 75};
   int i;

   // pass array's element pointers or addresses
   for(i=0;i<=2;i++)
   Largest(&A[i], &B[i]);
}

void Largest(int hold_arr1[ ], int hold_arr2[ ])
{
   int i =0, largest;

   if(hold_arr1[i] > hold_arr2[i])
      largest = hold_arr1[i];
   else
      largest = hold_arr2[i];
```

```
    printf("The largest is %d\n", largest);
}
```

```
The largest is 45
The largest is 70
The largest is 75
Press any key to continue . .
```

17. Next, re-write Largest() and main(). Have main() pass the entire array to Largest() and have Largest() go through a loop, printing the largest in each pair. Remember that when passing entire array, no subscript are provided.

```
#include <stdio.h>

void Largest(int [ ], int [ ]);

void main(void)
{
   int A[3] = {40, 70, 20}, B[3]={45, 35, 75};

   // pass array's first element pointers or addresses
   Largest(A, B);
}

void Largest(int hold_arr1[ ], int hold_arr2[ ])
{
   int i, largest;

   for(i=0;i<=2;i++)
   {
      if(hold_arr1[i] > hold_arr2[i])
         largest = hold_arr1[i];
      else
         largest = hold_arr2[i];
      printf("The largest is %d\n", largest);
   }
}
```

```
The largest is 45
The largest is 70
The largest is 75
Press any key to continue .
```

18. Change only Largest() so that the array x[ ] will have the larger number of each respective pair. In this example, x[ ] would become: 30, 50, 60.

    main() passed A[ ] to Largest() and Largest() received that array into hold_arr1[ ] as an argument. No copy of the array is made in Largest(), so that hold_arr1[ ] occupies the same location in memory as A[ ] does. Here, Largest() altered hold_arr1[ ] so it would effectively alter A[ ] in main().

```
#include <stdio.h>

void Largest(int [ ], int [ ]);

void main(void)
{
   int A[3] = {40, 70, 20}, B[3]={45, 35, 75};

   // pass array's first element pointers or addresses
   Largest(A, B);
}

void Largest(int hold_arr1[ ], int hold_arr2[ ])
{
   int i, x[3], largest;

   for(i=0;i<=2;i++)
   {
      if(hold_arr1[i] > hold_arr2[i])
         largest = hold_arr1[i];
      else
         largest = hold_arr2[i];

      x[i] = largest;
      printf("The largest is %d\n", x[i]);
   }
}
```

```
The largest is 45
The largest is 70
The largest is 75
Press any key to continue .
```

19. Next, rewrite main() so that it will print out the contents of A[ ]. Show the output.

```
#include <stdio.h>

void Largest(int [ ]);

void main(void)
{
   int i, A[3] = {40, 70, 20}, B[3]={45, 35, 75};

   // pass array's first element pointers or addresses
   Largest(A);
   printf("\nI'm in main().\n");
   for(i=0;i<=2;i++)
      printf("A[%d] = %d\n", i, A[i]);
}

void Largest(int x[ ])
{
   int i;

   printf("I'm in Largest().\n");
   for(i=0;i<=2;i++)
```

```
    {
        x[i] = x[i] + 5;
        printf("x[%d] = %d\n", i, x[i]);
    }
}
```

```
I'm in Largest().
x[0] = 45
x[1] = 75
x[2] = 25

I'm in main().
A[0] = 45
A[1] = 75
A[2] = 25
Press any key to continue . .
```

20. Next, change the problem and show the trace and the arrays if “child”, “ren” are passed to a function named Cat(). A sample trace header is shown below.

i	j	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	y[0]	y[1]	y[2]	y[3]
		'c'	...	...			'\0'				'r'	..	..	
0														
1														
.														
.														
.														
5														
	0						'r'							
	1							'e'						
	2								..					
	3									...				

Ans:

```
void Cat(char x[ ], char y[ ])
{
    int i, j;
    for(i = 0; x[i] != '\0'; i++)
        ; // this is a valid statement, an empty for statement

    for(j = 0; y[j] != '\0'; j++)
        x[i + j] = y[j];
}
```

i	j	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	y[0]	y[1]	y[2]	y[3]
		'c'	'h'	'i'	'l'	'd'	'\0'				'r'	'e'	'n'	'\0'
0														
1														
.														
.														
.														
5														
	0						'r'							
	1							'e'						
	2								'n'					
	3									'\0'				

21. In main(), will the first or the second string that is passed be altered? **Ans:** The function alters the first string so that the first string passed in main() will be altered. The second string stays the same.
22. In question 20, which loop, the first or the second, looks for the end of the first string? Which loop looks for the end of the second string? Which loop copies characters from one string to the end of the other? **Ans:** The first loop looks for the end of the first string by searching for the null character '\0'. The second loop looks for the end of the second string. The second loop copies characters from the second string to the end of the first one.

21. Write a function called FindLen() that will print the length of a string. For example, if main() called FindLen() by passing “Children”, then it will print the integer 8.

```
void FindLen(char x[ ])
{
    int i;
    for(i=0;x[i]!='\0';i++)
        ;
    printf("Length of %s is %d\n", x, i);
}
```

24. Complete the following code and show the output.

```
#include <stdio.h>

void main(void)
{
    int i = 2, A[3] = {5, 9, 4};
    DrillIt(i, A);
    printf("i = %d, A[1] = %d\n", i, A[1]);
}

void DrillIt(int j, int B[ ])
{
    j = 0;
    B[1] = 0;
}
```

```
#include <stdio.h>

void DrillIt(int, int [ ]);

void main(void)
{
    int i = 2, A[3] = {5, 9, 4};
    DrillIt(i, A);
    printf("i = %d, A[1] = %d\n", i, A[1]);
}

void DrillIt(int j, int B[ ])
{
    j = 0;
    B[1] = 0;
}
```

```
i = 2, A[1] = 0
Press any key to continue . .
```

Well, we have already come to the end of this worksheet. If you already noticed, all the programs

used functions that don’t have return value. In next worksheet we will learn functions that having return values.

www.tenouk.com