To:
Tenouk

C LAB WORKSHEET 12
C & C++ Functions Part 1 (with no return values)

1. Understanding the fundamental of and using C/C++ User-defined Functions.
2. Functions and loops.
3. 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().

Functions are one of the most important construct in C/C++. You may find other term for functions in other programming languages such as routines and procedures. Functions enable C/C++ programs built in structured manner. Each function will perform specific task. Built and tested functions can be reusable. Common tasks such as printing to the standard output and reading from standard input can be created and reused.
Till now you already used the built in or predefined function such as printf() and scanf()/scanf_s().
In this worksheet you will learn about user defined functions that is function created by users.
main() is also function with an exception that it is needed in all C/C++ program as an execution point. Program execution start at main() function.
Function may receive data and/or may also return data. The received data generally called **parameter** or placeholder where as the **actual value** stored in the placeholder normally called **argument**.

For example, consider the following main() program:

```
// start with main() function...
void main()
{
     // calling other functions...
     InitializeSettings();
     ReadInData();
     ProcessData();
     PrintReport();
}
```

We can see that this program is divided into four parts and we can get a general idea of what those parts accomplish. If we are concerned about a specific part, we can look at the code for that function and obtain the details. If we don't care, then the coding for that function is not cluttering up main().
After the initial investment to write and test the function, it can be called over and over again with only one line of code. The logic appears in only one place rather in several places in the large program so the code troubleshooting or tweaking and maintenance is easier.
Very large programs can be written by a team of programmers rather than by just one person and thus the job can be done faster. Each programmer would be assigned certain functions to write. Of course, this arrangement requires that what each function does, what arguments it received and what it returns must be documented accurately. Otherwise, there would be misunderstandings between programmers and how they write the functions. A complete story about main() can be found in Module Y.

Some Fundamental Definition

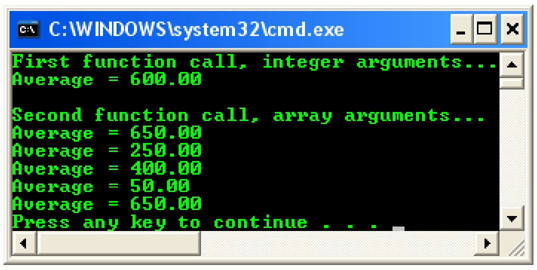
The function that is calling another one is called the "**calling function**" or **caller**. The function that is being called is called "**called function**" or "**callee**". Functions **may receive** arguments (through parameters) or they may not receive any. They may return only one value or none at all. If an array is being passed by the calling function, the called function may change it and the change could affect the original array in the calling function. This is not true of **scalars** (non arrays).

Example

Create an empty win32 console application named **mypointer**. Add C++ source file named **mypointersrc** and set up your project to be **compiled as C code**. Build and run the following program.

```
#include <stdio.h>

// prototype
void BalanceAvg(int, int);

// main() function
void main(void)
{
     // main() local variables...
     int Ball1[5] = {500, 200, 400, 100, 700}, // balance for one month
     Ball2[5] = {800, 300, 400, 0, 600},       // balance for another...
     i;    // an array's index

     // calling the function
     printf("First function call, integer arguments...\n");
     BalanceAvg(1000, 200);
     // ++i is same as i = i + 1
     printf("\nSecond function call, array arguments...\n");
     for(i = 0; i <= 4; ++i)
```

```
    // calling the function in a loop
    BalanceAvg(Ball1[i], Ball2[i]);
}

// BalanceAvg() will print the average of two
// integers with two decimal points,
// to call this function we pass two integers,
// upon completion, no return statement is executed, but the average of
// the integers will be printed. All task completed in function body...
void BalanceAvg(int x, int y)
{
    // BalanceAvg() local variable...
    float Average;

    // multiplying by 1.0 makes it a float
    // may generate warning, double to float...
    Average = ((x * 1.0) + y) / 2;
    printf("Average = %.2f\n", Average);
}
// after this, back to main...
```



```
First function call, integer arguments...
Average = 600.00

Second function call, array arguments...
Average = 650.00
Average = 250.00
Average = 400.00
Average = 50.00
Average = 650.00
Press any key to continue . . .
```

This example presents a very simple function. Aside from writing main(), this is the first time we are writing our own function. We have used printf() and others but they were predefined or system function that coming together with any compiler. The name of the user defined function is BalanceAvg(). This name appears in the program 4 times. It appears first in the function prototype, just before main(). Here void means that this function doesn't return something and the two int's inside the parentheses mean that the function requires two integers as receiving arguments. This is where the function is said to be declared. If the function is called or defined incorrectly, then this line would signal an inconsistency.
The function name appears the second and third time in main() when BalanceAvg() is called. At that time, two integers are passed as arguments.

```
BalanceAvg(1000, 200);
```

The first time, two numbers or constants are provided and the next time, numbers from the two arrays are provided, pair by pair, in a loop.

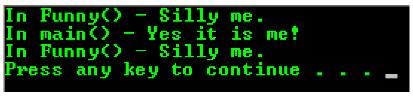
```
BalanceAvg(Ball1[i], Ball2[i]);
```

Ball1[ ], Ball2[ ] and i are local variable only to main(). BalanceAvg() isn't aware of their existence. Similarly, x, y and Average are local variables only to BalanceAvg() and main() isn't aware of their existence. In fact, if x were also named as Ball1 in BalanceAvg(), the program would execute the same as before. In that case, neither Ball1 would know that the other Ball1 existed.
We have two initialized arrays that contain the bank balance of five customers. Each array has the balances of the customers at the end of two months. First, we want to find the average of a new customer whose balances are not in the arrays. We pass 100 and 200 as arguments as we call the function BalanceAvg(). Now we are executing the code in this function.
Here x becomes 100 and y becomes 200, Average is calculated by first multiplying x by 1.0.
This makes the answer a floating point value, so the average is calculated to be 600.0 and then it is printed. The function definition shows void, meaning that this function doesn't return anything, and it doesn't. Control of execution comes back to main(), where it was left off. The for loop is next.
In the for loop, i varies from 0 to 4. Each time we call the function we pass the next pair of balances to the function, where the average is printed. In this loop, the function is called five times and we return from the function five times. Whenever we return from the function, the value of i is remembered so the loop knows how many more times to iterate. Once the loop is over, main() is complete.

Experiments

1. Build, run and show the output for the following example. Answer the questions.

```
#include <stdio.h>

// this is function prototype, must appear before function definition...
void Funny(void);
// main() function
void main(void)
{
  // main() calls Funny()
  Funny();
  printf("In main() - Yes it is me!\n");
}

// defining Funny...
void Funny(void)
{
  printf("In Funny() - Silly me.\n");
}
```



```
In Funny() - Silly me.
In main() - Yes it is me!
In Funny() - Silly me.
Press any key to continue . . .
```

a. Name two user-defined functions (function bodies which you have typed)?
b. Name one predefined/system function (a function which is made available to you by the compiler).
c. Execution starts at main(). What is the first function that main() calls?
d. What is the second function that main() calls after the first one is done?
e. What function does Funny() call?
f. Number the following events in order from 1 through 3:

______ - Funny() calls printf()

______ - main() calls printf()
______ - main() calls Funny()

g. How do we know where the definition of main() starts and ends?
h. How do we know where the definition of Funny() starts and ends?
i. The void after main() indicates that main() doesn't receive any arguments. Does Funny() receive any values?
j. The void before main() indicates that the main() doesn't return any values. Does Funny() return any values?
k. All programs must have main(). However, when defining a function other than main(), a prototype should be given, stating the name of the function, what arguments it receives and what item it returns, if any. A prototype can be given inside or outside main() provided that the function definition appear after the prototype statement. Where is the prototype given here?

Ans:

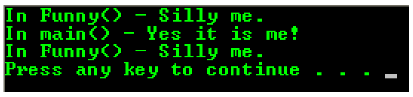
a. main() and Funny().
b. printf().
c. Funny().
d. printf().
e. printf().
f. …

__**2**__- Funny() calls printf()
__**3**__- main() calls printf()
__1__- main() calls Funny()

g. Starts from opening brace, { after the main function header, main() and ends with the matched closing brace, }.
h. Starts from the opening brace, { after the Funny function header, void Funny(void) and ends with the matched closing brace }
i. No it doesn't because of the void keyword.
j. NO it doesn't because of the void keyword.
k. At the top of program after the #include preprocessor directive.

2. Add another Funny() after the printf() in main(). Build, run and show the output. Answer the questions that follow.

```
#include <stdio.h>

// this is function prototype
void Funny(void);

// main()function
void main(void)
{
   // main() calls Funny()
   Funny();
   printf("In main() – Yes it is me!\n");
   // another call…
   Funny();
}

// defining Funny…
void Funny(void)
{
   printf("In Funny() – Silly me.\n");
}
```



```
In Funny() - Silly me.
In main() - Yes it is me!
In Funny() - Silly me.
Press any key to continue . . .
```

a. Number the events in order from 1 to 5.

_____ - Funny() calls printf() first time
_____ - Funny() calls printf() second time
_____ - main() calls printf()
_____ - main() calls Funny()
_____ - main() calls Funny()

b. When main() calls Funny(), which function do you think is the calling function?
c. Which function do you think is the called function?
d. When Funny() calls printf(), which function do you think is the calling function?
e. Which function do you think is the called function?
f. Which function gets called and also calls another function?
g. When printf() is called from main(), "Yes it is me!\n" is the argument passed to printf(). What argument is being passed to printf() from Funny()?
h. When main() calls Funny(), is there an argument passed to Funny()?

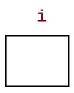
Ans:

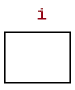
a. __2__- Funny() calls printf() first time
__5__- Funny() calls printf() second time
__3__- main() calls printf()
__4__- main() calls Funny()
__1__- main() calls Funny()

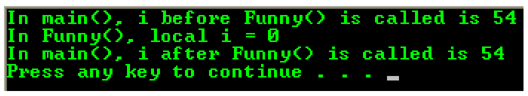
b. main() is the calling function or caller.
c. Funny() is called function or callee.
d. Funny() is calling function or caller.
e. printf() is called function or callee.
f. Funny() gets called and also calls another function, that is printf().
g. The argument passed is "In Funny() – Silly me.\n".
h. No, there is no argument passed to Funny(), the void keyword should be used here, void Funny(void).

3. Next run the following program, show the output and answer the questions.

```
#include <stdio.h>

// This is function prototype
void Funny(void);

// main()function
void main(void)
{
   int i = 54;

   printf("In main(), i before Funny() is called is %d\n", i);
   Funny();
   printf("In main(), i after Funny() was called is %d\n", i);
}

// Defining Funny...
void Funny(void)
{
   int i = 0;

   printf("In Funny(), local i = %d\n", i);
   i = -28;
}
```







```
In main(), i before Funny() is called is 54
In Funny(), local i = 0
In main(), i after Funny() is called is 54
Press any key to continue . . .
```

i = 54 in the first box (top) and i = 0 in the second box (bottom).

a. i is a local variable in main(), which means that only main() has access to it. What is the local variable in Funny()?
b. main() sets the value of i to 54. Then it was printed. Execution then passed to Funny(). At the beginning of Funny() was the value of i equal to 54 or just "garbage"?
c. Funny() set the value of i to -28. After coming back to main(), what was the value of i?
d. Was the i in main() the same as i in Funny()?

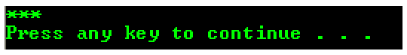
Ans:

a. Also an i however this i is different from the i in the main() though they have same name. Both i have different scope and class storage, that is they are stored at different memory location with same name.
b. From the program output sample, the value of i is 54.
c. This i value is -28.
d. No they are different.

4. Show the output for the following codes and answer the questions.

4(a)

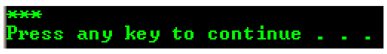
```
#include <stdio.h>

void Funny(void);

void main(void)
{
   Funny();
}

void Funny(void)
{
   int i;
   for(i = 1; i <=3; i = i + 1)
         printf("*");
   printf("\n");
}
```



```
***
Press any key to continue . . .
```

4(b)

```
#include <stdio.h>

void Funny(void);

void main(void)
{
   int i;
   for(i = 1; i <=3; i = i + 1)
         Funny();
   printf("\n");
}
```



```
***
Press any key to continue . . .
```

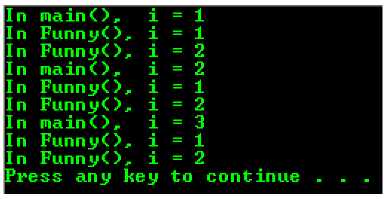
```
void Funny(void)
{
  printf("*");
}
```

a. How many times was Funny() called in Experiment 4(a)? In Experiment 4(b).
b. Every time that Funny() was called, printf() was called how many times in Experiment 4(a)? In experiment 4(b)?
c. In total, how many times was printf() called in Experiment 4(a)? In Experiment 4(b).

a. Once in 4(a) and 3 times in 4(b).
b. 4 times in 4(a) and 3 times in 4(b).
c. 4 times in 4(a) and 4 in 4(b).

5. Build, run and show the output for the following program. Then answer the questions.

```
#include <stdio.h>

void Funny(void);

void main(void)
{
  int i;
  for(i = 1; i <=3; i = i + 1)
  {
       printf("In main(),  i = %d\n", i);
       Funny();
  }
}

void Funny(void)
{
  int i;
  for(i = 1; i <= 2; i = i + 1)
       printf("In Funny(), i = %d\n", i);
}
```



```
In main(),  i = 1
In Funny(), i = 1
In Funny(), i = 2
In main(),  i = 2
In Funny(), i = 1
In Funny(), i = 2
In main(),  i = 3
In Funny(), i = 1
In Funny(), i = 2
Press any key to continue . . .
```

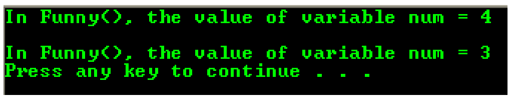
a. How many times does main() call Funny()?
b. Each time that Funny() is called, printf() is called how many times from Funny()?
c. The loop in main() is done how many times?
d. The loop in Funny() is done a total of how many times?
e. The printf() in main() is done how many times?
f. What is the total number of times that printf() is called from Funny()?
g. Tracing the execution of this program, initially, main() sets i to 1. Then the control of execution goes to Funny(). Here, i is also set to 1. From the output, can you determine whether this is the same i, or are they different?
h. Starting again, the i in main() is set to 1 and then the i in Funny() goes from 1 to what value? After that, we go back to main() and now i is incremented to 2 and we come back to Funny(). When we come back to Funny(), i goes from 1 to what value? The third time we come back to Funny(), i goes from 1 to what value?

Ans:

a. 3 times.
b. 2 times.
c. 3 times.
d. 2 times.
e. 3 times.
f. 2 times.
g. They are different. Both i are local to their functions.
h. i in Funny() goes from 1 to 2. From 1 to 2. Also from 1 to 2. So, the total is 6 depicted by the "In Funny(), i = #" lateral string print out in the output.

6. Let us pass **arguments** (values or data) to Funny(). The difference for this thing when a function receives an argument are shown in bold, an
integer represented by **parameter num** instead of void.

```
#include <stdio.h>

// receiving an integer
void Funny(int);

void main(void)
{
  // first call, 4 will go into num
  Funny(4);
  printf("\n");
  // second call, 3 will go into num
  Funny(3);
}

// num is a formal parameter, acts as a placeholder.
// the value or argument of num can change...
void Funny(int num)
{
  printf("In Funny(), the value of variable num = %d\n", num);
```



```
In Funny(), the value of variable num = 4

In Funny(), the value of variable num = 3
Press any key to continue . . .
```

```
}
```

a. main() calls Funny() how many times?
b. What is the difference in how Funny() is called here compared to how it is called in Experiment 5.
c. When Funny() is called by main() for the first time, what argument (an integer) is passed?
d. When Funny() is called by main() for the second time, what argument is passed?
e. What is the value of num when Funny() is executed the first time?
f. What is the value of num when Funny() is executed the second time?
g. In the **prototype** for Funny(), should the data type of the argument be given?
h. In the prototype for Funny(), do we have to give the variable name of the argument, that is, num?
i. In the **definition** for Funny(), should we give the data type of the argument?
j. In the definition for Funny(), is the variable name of the argument given?
k. Number the order of the events execution:

_______ - Funny() starts executing with num equal to 3
_______ - Funny() starts executing with num equal to 4.
_______ - main() calls printf() and sends one format string as an argument.
_______ - main() calls Funny() and sends 3 as an argument.
_______ - main() calls Funny() and sends 4 as an argument.
_______ - Funny() sends two arguments to printf(), a format string and num, which has a value of 3.
_______ - Funny() sends two arguments to printf(), a format string and num, which has a value of 4.

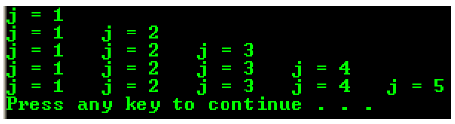
Ans:

a. 2 times.
b. Funny() is called with an argument in experiment 6.
c. 4 was passed.
d. 3 was passed.
e. An integer 4.
f. An integer 3.
g. Yes it is a must.
h. No, it is an optional.
i. Yes it is a must and this data type must match with the data type declared in prototype. In this case it is a int.
j. Yes it is given, a num.
k. ...

__**7**__- Funny() starts executing with num equal to 3
__**3**__- Funny() starts executing with num equal to 4.
__**4**__- main() calls printf() and sends one format string as an argument.
__**5**__- main() calls Funny() and sends 3 as an argument.
__**1**__- main() calls Funny() and sends 4 as an argument.
__**6**__- Funny() sends two arguments to printf(), a format string and num, which has a value of 3.
__**2**__- Funny() sends two arguments to printf(), a format string and num, which has a value of 4.

7. Try the following program. Show the output and answer the questions.

```
#include <stdio.h>

// function prototype
void Funny(int);

void main(void)
{
   int i;
   for(i = 1; i <= 5; i = i + 1)
         Funny(i);
}

// function definition
void Funny(int num)
{
   // local variable, local to Funny()
   int j;
   for(j = 1; j <= num; j = j + 1)
         printf("j = %d\t", j);
   printf("\n");
}
```



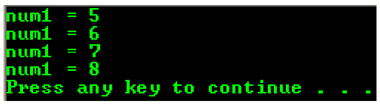
```
j = 1
j = 1   j = 2
j = 1   j = 2   j = 3
j = 1   j = 2   j = 3   j = 4
j = 1   j = 2   j = 3   j = 4   j = 5
Press any key to continue . . .
```

a. If j were also named i in Funny(), would there have been a problem?
b. main() calls Funny() how many times?
c. Each time that Funny() was called, what were the values that were passed?
d. What were the values of num each time Funny() was called?
e. For every time that Funny() was called, how many times was the j loop executed?

a. No there should be no problem because both i are local to their respective function. One is local to main() function and another is local to Funny().
b. 5 times based on the for loop in main().
c. The values passed are 1, 2, 3, 4 and 5.
d. The values of num are 1, 2, 3, 4 and 5.
e. Up to and inclusive the value of num and they are 1, 2, 3, 4, and 5.

8. Try next program. Show the output and answer the questions.

```
#include <stdio.h>

// function prototype
void Funny(int, int);

void main(void)
{
   Funny(5, 8);
}

// function definition
void Funny(int num1, int num2)
{
   for( ; num1 <= num2; num1 = num1 + 1)
         printf("num1 = %d\n", num1);
}
```



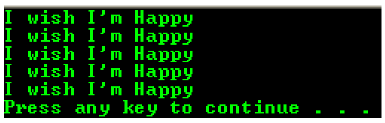
```
num1 = 5
num1 = 6
num1 = 7
num1 = 8
Press any key to continue . . .
```

a. When two arguments are passed to a function, should data types be given for each one in the **prototype**?
b. In the definition of Funny(), do we need the keyword, int in front of num2 as well? Try it.
c. main() passes what two arguments to Funny()?
d. In Funny(), how do we know which of the passed values will become num1 and num2?
e. How many times does main() call Funny()? How many times does Funny() call printf()?

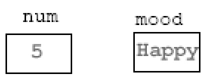
a. Yes it is a must.
b. Yes it is a must.
c. 5 and 8.
d. It is in order. That is the first argument will be passed to the first parameter and so on. In this case 5 will be copied and stored in num1 and 8 will be copied and stored in num2.
e. main() calls Funny() once and Funny() calls printf() 4 times.

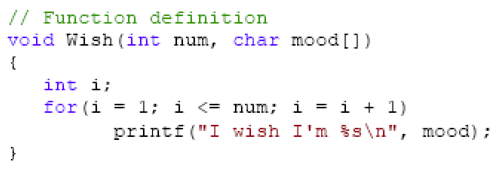
9. In the following program, let pass an array to a function. Show the output and answer the questions.

```
#include <stdio.h>

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

void main(void)
{
   Wish(5, "Happy");
}
```

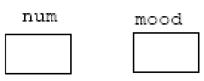


```
I wish I'm Happy
I wish I'm Happy
I wish I'm Happy
I wish I'm Happy
I wish I'm Happy
Press any key to continue . . .
```





```
// Function definition
void Wish(int num, char mood[])
{
    int i;
    for(i = 1; i <= num; i = i + 1)
          printf("I wish I'm %s\n", mood);
}
```



a. What are the two values passed to Wish()?
b. In Wish(), what becomes the value of num? What becomes the value of mood? Write them in the boxes provided.
c. Notice that in both the **prototype** and the **definition** of Wish(), there is no number in the brackets, giving the number of cells of mood[ ]. Can you think why omitting this number makes the function more flexible for use in other instances?
d. How can we tell that “Happy” is a character string? How can we tell that mood[ ] should also be a character string?
e. If Wish() were called from main() with this statement, Wish(3, "Excited"); , then what would be the output?

a. An integer 5 and a string "Happy".
b. num is an integer of 5 and mood is a string of "Happy".
c. This unsized array make the system decide the actual size needed for storage.
d. "Happy" is a character string because it is enclosed in the double quotes and an array mood[ ] has been declared as a char type.
e. Only the first 3 alphabets from "Excited" will be displayed that is "I wish I'm Exc".