Bilkent University

Department of Computer Engineering

CS 315

Programming Language

Homework 2

Subprogram Parameters

24.04.2018

Fuad Aghazada: 21503691

Section 1

Instructor:

- Halil Altay Güvenir

Table of contents

Example Program in 5 different languages

1. Python

1.1 Code

```
'''
    Analyzing function parameters in Python

    @author: Fuad Aghazada
    @date: 20/04/2018
'''

# Example function for demonstrating positional and keyword parameter c.
def myFunction0(par0, par1):
    print(str(par0) + " is the first parameter!")
    print(str(par1) + " is the second parameter!\n")

# Example function for demonstrating default parameter values
def myFunction1(par0, par1 = "I am a default parameter", par2 = "Me too", par3 = 100):
    print("Parameter 0: " + str(par0))
    print("Parameter 1: " + str(par1))
    print("Parameter 2: " + str(par2))
    print("Parameter 3: " + str(par3) + "\n")

# Example function for demonstrating variable number of parameters
def myFunction2(*pars):

    # printing number of parameters
    print("Number of parameters: " + str(len(pars)))

    # printing all parameters
    for p in pars:
        print(p)

    print("\n")
```

```
# Example function for demonstrating parameter passing
def myFunction3(par0, par1, par2):

    # modifying the values of the parameters
    par0 += 77
    par1 = "New String"
    par2.append(100)

    # Values of parameters inside the function
    print("In myFunction3 par0: " + str(par0))
    print("In myFunction3 par1: " + str(par1))
    print("In myFunction3 par2: " + str(par2) + "\n")


# Positional and Keyword parameter correspondence
myFunction0("sampleString", 7)                    # positional arguments
myFunction0(par0 = "sampleString", par1 = 7)      # keyword arguments
myFunction0(par1 = 7, par0 = "sampleString")      # keyword arguments

# Formal parameter default values
myFunction1("sampleString")
myFunction1("sampleString", 7)
myFunction1("sampleString", 7, "new_par_3")
myFunction1("sampleString", par2 = 7)
myFunction1("sampleString", par3 = "new_par_3")

# Variable number of parameters
myFunction2("myPar")
myFunction2(1, 2, 3, "myPar")

# Parameter passing
first_par = 7
second_par = "sampleString"
third_par = [1, 2, "aString"]

# Values before myFunction3
print("Before calling myFunction3 first_par: " + str(first_par))
print("Before calling myFunction3 second_par: " + str(second_par))
print("Before calling myFunction3 third_par: " + str(third_par) + "\n")

myFunction3(first_par, second_par, third_par)     # calling myFunction3


# Values after myFunction3
print("After calling myFunction3 first_par: " + str(first_par))
print("After calling myFunction3 second_par: " + str(second_par))
print("After calling myFunction3 third_par: " + str(third_par) + "\n")
```

1.2 Output

```
sampleString is the first parameter!
7 is the second parameter!

sampleString is the first parameter!
7 is the second parameter!

sampleString is the first parameter!
7 is the second parameter!

Parameter 0: sampleString
Parameter 1: I am a default parameter
Parameter 2: Me too
Parameter 3: 100

Parameter 0: sampleString
Parameter 1: 7
Parameter 2: Me too
Parameter 3: 100

Parameter 0: sampleString
Parameter 1: 7
Parameter 2: new_par_3
Parameter 3: 100

Parameter 0: sampleString
Parameter 1: I am a default parameter
Parameter 2: 7
Parameter 3: 100

Parameter 0: sampleString
Parameter 1: I am a default parameter
Parameter 2: Me too
Parameter 3: new_par_3

Number of parameters: 1
myPar


Number of parameters: 4
1
2
3
```

```
myPar


Before calling myFunction3 first_par: 7
Before calling myFunction3 second_par: sampleString
Before calling myFunction3 third_par: [1, 2, 'aString']

In myFunction3 par0: 84
In myFunction3 par1: New String
In myFunction3 par2: [1, 2, 'aString', 100]

After calling myFunction3 first_par: 7
After calling myFunction3 second_par: sampleString
After calling myFunction3 third_par: [1, 2, 'aString', 100]
```

1.3 Explanation

Python is a scripting language which has a subprogram with both positional and keyword correspondence so that like in the example function of myFunction0 can be called either entering the parameters in the same order that they were defined or assigning the values to the parameter names.

Python subprograms have also the feature of assigning default values to the parameters like in the example of myFunction1. The problem here is that we cannot have a non-default parameter following a default parameter, otherwise Python 3 interpreter returns an error message.

We can also use variable number of parameters in Python subprograms by adding ‘*’ in front of the parameter name. In this case this variable will keep all the parameters entered in itself like a list so that we can have an access to its length and the elements in it like in the example of myFunction2.

Parameter passing method of Python is different from some popular imperative languages to that it uses ‘pass-by-object-reference’. In this kind of parameter passing method Python uses pass-by-value when the primitive data types are passed, otherwise uses pass-by-reference when the passing data is mutable type. Examples for immutable types are integer, string, double and etc. The example for mutable type is the list in Python. In the example of myFunction3, the passing of integer, string (immutable) and lists (mutable) are described.

2. JavaScript

2.1 Code

```
<script type = "text/javascript">

   /**
      Analyzing function parameters in JavaScript

      @author: Fuad Aghazada
      @date: 20/04/2018
   */

   // Example function for demonstrating positional / keyword parameter c.
   function myFunction0(par0, par1)
   {
      document.write(par0 + " is the first parameter <br>");
      document.write(par1 + " is the second parameter <br><br>");
   }

   // Example function for demonstrating default parameter value
   function myFunction1(par0, par1 = "I am a default parameter", par2 = "Me too", par3 =
100)
   {
      document.write("Parameter 0: " + par0 + "<br>");
      document.write("Parameter 1: " + par1 + "<br>");
      document.write("Parameter 2: " + par2 + "<br>");
      document.write("Parameter 3: " + par3 + "<br><br>");
   }

   // Example function for demonstrating variable number of parameters
   function myFunction2()
   {
      // printing number of parameters
      document.write("Number of parameters: " + arguments.length + "<br>");

      // printing all parameters
      for( var i = 0; i < arguments.length; i++)
      {
         document.write(arguments[i] + "<br>");
      }
      document.write("<br>");
   }

   // Example function for parameter passing
   function myFunction3(par0, par1, par2)
   {
    // modifying the values of the parameters
    par0 += 77
    par1 = "New String"
    par2.push(100)

    // Values of paramaters inside the function
    document.write("In myFunction3 par0: " + par0 + "<br>");
    document.write("In myFunction3 par1: " + par1 + "<br>");
    document.write("In myFunction3 par2: " + par2 + "<br><br>")
}

// Positional / Keyword parameters
myFunction0("sampleString", 7);
myFunction0(par0 = "sampleString", par1 = 7);
myFunction0(par1 = 7, par0 = "sampleString");

// Formal parameter default values
myFunction1("sampleString");
myFunction1("sampleString", 7);
myFunction1("sampleString", 7, "new_par_3");
myFunction1("sampleString", par2 = 7);
myFunction1("sampleString", par3 = "new_par_3");

// Variable number of parameters
myFunction2("myPar");
myFunction2(1, 2, 3, "myPar");

// Parameter passing
var first_par = 7;
var second_par = "sampleString";
var third_par = [1, 2, "aString"];

// Values before myFunction3
document.write("Before calling myFunction3 first_par: " + first_par + "<br>");
document.write("Before calling myFunction3 second_par: " + second_par + "<br>");
document.write("Before calling myFunction3 third_par: " + third_par + "<br><br>");

myFunction3(first_par, second_par, third_par);

// Values after myFunction3
document.write("After calling myFunction3 first_par: " + first_par + "<br>");
document.write("After calling myFunction3 second_par: " + second_par + "<br>");
document.write("After calling myFunction3 third_par: " + third_par + "<br><br>");

</script>
```

2.2 Output

```
sampleString is the first parameter
7 is the second parameter

sampleString is the first parameter
7 is the second parameter

7 is the first parameter
sampleString is the second parameter

Parameter 0: sampleString
Parameter 1: I am a default parameter
Parameter 2: Me too
Parameter 3: 100

Parameter 0: sampleString
Parameter 1: 7
Parameter 2: Me too
Parameter 3: 100

Parameter 0: sampleString
Parameter 1: 7
Parameter 2: new_par_3
Parameter 3: 100

Parameter 0: sampleString
Parameter 1: 7
Parameter 2: Me too
Parameter 3: 100

Parameter 0: sampleString
Parameter 1: new_par_3
Parameter 2: Me too
Parameter 3: 100

Number of parameters: 1
myPar

Number of parameters: 4
1
2
3
myPar

Before calling myFunction3 first_par: 7
Before calling myFunction3 second_par: sampleString
Before calling myFunction3 third_par: 1,2,aString

In myFunction3 par0: 84
In myFunction3 par1: New String
In myFunction3 par2: 1,2,aString,100

After calling myFunction3 first_par: 7
After calling myFunction3 second_par: sampleString
After calling myFunction3 third_par: 1,2,aString,100
```

2.3 Explanation

JavaScript has also similar points on subprogram features with Python so that it also has both positional and keyword parameter correspondence like in the example function of myFunction0.

JavaScript subprograms have default parameter feature similar to Python like in the example function of myFunction1.

For variable number of parameters JavaScript uses ‘arguments’ array for accessing all the parameter entered for the function like in the example of myFunction2.

Parameter passing method is also similar to Python so that when you enter passing parameter as integer, string and etc. it does not change (immutable), however when the parameter is an array (mutable) the method changes the content of it.

3. PHP

3.1 Code

```
<?php
   /***
      Analyzing function parameters in PHP

      @author: Fuad Aghazada
      @date: 20/04/2018
   */

   // Example function for demonstrating positional and keyword parameter c
   function myFunction0($par0, $par1)
   {
      echo $par0." is the first parameter\n";
      echo $par1." is the second parameter\n\n";
   }

   // Example function for demonstrating default parameter correspondence
   function myFunction1($par0, $par1 = "I am a default parameter", $par2 = "Me too", $par3
= 100)
   {
      echo "Parameter 0: ".$par0."\n";
      echo "Parameter 1: ".$par1."\n";
      echo "Parameter 2: ".$par2."\n";
      echo "Parameter 3: ".$par3."\n\n";
   }

   // Example function for demonstrating variable number of parameters
   function myFunction2()
   {
      // printing number of parameters
      $number_of_pars = func_num_args();

      echo "Number of parameters: ".$number_of_pars."\n";

      for($i = 0; $i < $number_of_pars; $i++)
      {
         echo func_get_arg($i)."\n";
      }
      echo "\n";
   }
   // Example function for demonstrating parameter passing
   function myFunction3($par0, $par1, &$par2)
   {
     // modifying the values of the parameters
```

```
        $par0 += 77;
        $par1 = "New String";
        $par2 = array(4, 5, 6);

        // values of parameters inside the function
        echo "In myFunction3 par0: $par0 \n";
        echo "In myFunction3 par1: $par1 \n";
        echo "In myFunction3 par2: ";
        print_r($par2);
        echo "\n\n";
    }

    // Positional parameter correspondence -- Keyword is not available in PHP
    myFunction0("sampleString", 7);

    // Formal parameter default values
    myFunction1("sampleString");
    myFunction1("sampleString", 7);
    myFunction1("sampleString", 7, "new_par_3");

    // Variable number of parameters
    myFunction2("myPar");
    myFunction2(1, 2, 3, "myPar");

    // Parameter passing
    $first_par = 7;
    $second_par = "sampleString";
    $third_par = array(1, 2, "aString");

    // Values before myFunction3
    echo "Before calling myFunction3 first_par: $first_par \n";
    echo "Before calling myFunction3 second_par: $second_par \n";
    echo "Before calling myFunction3 third_par: ";
    print_r($third_par);
    echo "\n\n";

    myFunction3($first_par, $second_par, $third_par);          // calling myFunction3


    // Values after myFunction3
    echo "After calling myFunction3 first_par: $first_par \n";
    echo "After calling myFunction3 second_par: $second_par \n";
    echo "After calling myFunction3 third_par:";
    print_r($third_par);
    echo "\n\n";

?>
```

3.2 Output

```
sampleString is the first parameter
7 is the second parameter

Parameter 0: sampleString
Parameter 1: I am a default parameter
Parameter 2: Me too
Parameter 3: 100

Parameter 0: sampleString
Parameter 1: 7
Parameter 2: Me too
Parameter 3: 100

Parameter 0: sampleString
Parameter 1: 7
Parameter 2: new_par_3
Parameter 3: 100

Number of parameters: 1
myPar

Number of parameters: 4
1
2
3
myPar

Before calling myFunction3 first_par: 7
Before calling myFunction3 second_par: sampleString
Before calling myFunction3 third_par: Array
(
   [0] => 1
   [1] => 2
   [2] => aString
)


In myFunction3 par0: 84
In myFunction3 par1: New String
In myFunction3 par2: Array
(
   [0] => 4
   [1] => 5
   [2] => 6
)
```

```
After calling myFunction3 first_par: 7
After calling myFunction3 second_par: sampleString
After calling myFunction3 third_par:Array
(
   [0] => 4
   [1] => 5
   [2] => 6
)
```

3.3 Explanation

Unlike the previous script languages that I analyzed, PHP does not support keyword parameter correspondence, uses only formal parameters like in the example of myFunction0.

PHP subprograms have a feature of assigning default values to the parameters like in the example of myFunction1.

In order to use variable number of parameters in PHP, the user should use ‘func_num_args’ for the number of parameters entered for the function and to access them use ‘func_get_args’ array like in the example of myFunction2.

PHP have two different parameter passing methods; both ‘pass-by-value’ and ‘pass-by-reference’. When ‘pass-by-value’ is used the value of the passed variable does not change, however, when the ‘pass-by-reference’ is used, the value of the passed variable is modified. To use ‘pass-by-reference’ we need to put ‘&’ in front of the parameter name when the function is defined.

4. C++

4.1 Code

```
/**
   Analyzing function parameters in C++

   @author: Fuad Aghazada
   @date: 20/04/2018
*/
#include <iostream>
#include <string>
#include <stdarg.h>


using namespace std;

// Example function for demonstrating positional and keyword parameter c
void myFunction0(string par0, int par1)
{
   cout << par0 << " is the first parameter" << endl;
   cout << par1  << " is the second parameter\n" << endl;
}

// Example function for demonstrating default parameter correspondence
void myFunction1(string par0, int par1 = 666, string par2 = "Me too", int par3 = 100)
{
   cout << "Parameter 0: " << par0 <<  endl;
   cout << "Parameter 1: " << par1 <<  endl;
   cout << "Parameter 2: " << par2 <<  endl;
   cout << "Parameter 3: " << par3 << "\n" <<  endl;
}

// Example function for demonstrating variable number of parameters
void myFunction2(int parameters, ...)
{
   // variable list
   va_list valist;

   // creating variable list for the parameters
   va_start(valist, parameters);



   // Printing number of parameters
   cout << "Number of paramaters: " << parameters << endl;
```

```
    for(int i = 1; i <= parameters; i++)
    {
        cout << va_arg(valist, int) << endl;
    }

    cout << endl;

    // cleaning the memory
    va_end(valist);
}

// Example function for demonstrating parameter passing
void myFunction3(int par0, string par1, int *&par2, int **par3)
{
    // modifying the value of the parameters
    par0 += 77;
    par1 = "New String";

    par2[0] = 4;
    par2[1] = 5;
    par2[2] = 6;

    *(*par3) = 4;
    *(*par3 + 1) = 5;
    *(*par3 + 2) = 6;

    // values of parameters inside the function
    cout << "In myFunction3 par0: " << par0 << endl;
    cout << "In myFunction3 par1: " << par1 << endl;
    cout << "In myFunction3 par2: " << par2[0] << ", " << par2[1] << ", " << par2[2] << endl;
    cout << "In myFunction3 par3: " << *(*par3) << ", " << *(*par3 + 1) << ", " << *(*par3 + 2)
<< "\n" << endl;
}




// Example function for demonstrating variadic templates
void myFunction4()
{
    return;         // this function is the case when the parameters are empty
}

template <typename variadic, typename... parameters>
variadic myFunction4(variadic par, parameters... pars)
{
    // printing the parameters
    cout << par << endl;

    // recursive call for the remaining parameters
    myFunction4(pars...);
}

// Main function for execution
int main()
{
    // Positional parameter correspondence -- Keyword is not available in C++
    myFunction0("sampleString", 7);

    // Formal parameter default values
    myFunction1("sampleString");
    myFunction1("sampleString", 7);
    myFunction1("sampleString", 7, "new_par_3");

    // Parameter passing
    int first_par = 7;
    string second_par = "sampleString";
    int *third_par = new int[3];
    third_par[0] = 1;
    third_par[1] = 2;
    third_par[2] = 3;

    int *forth_par = new int[3];
    forth_par[0] = 1;
    forth_par[1] = 2;
    forth_par[2] = 3;

    // Values before myFunction3
    cout << "Before calling myFunction3 first_par: " << first_par << endl;
    cout << "Before calling myFunction3 second_par: " << second_par << endl;
    cout << "Before calling myFunction3 third_par: " << third_par[0] << ", " << third_par[1] <<
", " << third_par[2] << endl;
    cout << "Before calling myFunction3 forth_par: " << forth_par[0] << ", " << forth_par[1] <<
", " << forth_par[2] << "\n" << endl;

    myFunction3(first_par, second_par, third_par, &forth_par);

    // Values after myFunction3
    cout << "After calling myFunction3 first_par: " << first_par << endl;
    cout << "After calling myFunction3 second_par: " << second_par << endl;
    cout << "After calling myFunction3 third_par: " << third_par[0] << ", " << third_par[1] << ", "
<< third_par[2] << endl;
```

```
    cout << "After calling myFunction3 forth_par: " << forth_par[0] << ", " << forth_par[1] << ",
" << forth_par[2] << "\n" << endl;

    // deleting the array from heap
    delete third_par;
    delete forth_par;

    // Variadic templates
    myFunction4(1, 2, 12, 2342);
    myFunction4("string1", "string2", "string3");

    return 0;
}
```

4.2 Output

```
sampleString is the first parameter
7 is the second parameter

Parameter 0: sampleString
Parameter 1: 666
Parameter 2: Me too
Parameter 3: 100

Parameter 0: sampleString
Parameter 1: 7
Parameter 2: Me too
Parameter 3: 100

Parameter 0: sampleString
Parameter 1: 7
Parameter 2: new_par_3
Parameter 3: 100

Number of paramaters: 1
1

Number of paramaters: 3
1
2
3


Before calling myFunction3 first_par: 7
Before calling myFunction3 second_par: sampleString
Before calling myFunction3 third_par: 1, 2, 3
Before calling myFunction3 forth_par: 1, 2, 3

In myFunction3 par0: 84
In myFunction3 par1: New String
In myFunction3 par2: 4, 5, 6
In myFunction3 par3: 4, 5, 6

After calling myFunction3 first_par: 7
After calling myFunction3 second_par: sampleString
After calling myFunction3 third_par: 4, 5, 6
After calling myFunction3 forth_par: 4, 5, 6

1
2
12
2342
string1
string2
string3
```

4.3 Explanation

C++ language unlike Python and JavaScript does not support keyword parameters, uses only positional parameter correspondence.

However, C++ also has a feature of keeping default values for the parameters like in the example of myFunction1.

By adding <stdargs.h> library, we can use variable number of parameters in C++ so that we will pass number of parameters and the parameters themselves as arguments to the function like in the example of myFunction2.

Parameter passing method is also similar to PHP by having both 'pass-by-value' and 'pass-by-reference'. For 'pass-by-reference' we can either use pointers by putting '*' or putting '&' in front of the parameter name like in the example of myFunction3.

C++ 11 has a new feature of variadic templates, with which the subprograms can take variable number of parameters without encountering with cumbersome by using va_list as in the example of myFunction2. In the example of myFunction4 variable number of parameters has been illustrated using variadic templates.

5. C

5.1 Code

```
/**
   Analyzing function parameters in C

   @author: Fuad Aghazada
   @date: 20/04/2018
*/

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

// Example function for demonstrating positional / keyword parameter correspondence
void myFunction0(char par0[], int par1)
{
   printf("%s is the first parameter\n", par0);
   printf("%d is the second parameter\n\n", par1);
}

// Example function for demonstrating variable number of parameters
void myFunction2(int parameters, ...)
{
   // variable list
   va_list valist;

   // creating variable list for the parameters
   va_start(valist, parameters);

   // Printing number of parameters
   printf("Number of paramaters: %d\n", parameters);

   int i;
   for(i = 1; i <= parameters; i++)
   {
      printf("%d\n", va_arg(valist, int));
   }

   printf("\n");

   // cleaning the memory
   va_end(valist);
}
```

```
// Example function for demonstrating parameter passing
void myFunction3(int par0, char par1[], int ** par2)
{
   // modifying the value of the parameters
   par0 += 77;
   par1 = "New String";

   *(*par2) = 4;
   *(*par2 + 1) = 5;
   *(*par2 + 2) = 6;

   // Values of parameters inside myFunction3
   printf("In myFunction3 par0: %d\n", par0);
   printf("In myFunction3 par1: %s\n", par1);
   printf("In myFunction3 par2: %d, %d, %d\n\n", *(*par2), *(*par2 + 1), *(*par2 + 2));
}


// Main function
int main()
{
   // Positional / Keyword parameter
   myFunction0("sampleString", 7);

   // Variable number of passing
   myFunction2(1, 1);
   myFunction2(3, 1, 2, 3);

   // Parameter passing
   int first_par = 7;
   char *second_par = "sampleString";
   int  *third_par = malloc(3 * sizeof(int));

   third_par[0] = 1;
   third_par[1] = 2;
   third_par[2] = 3;

   // Values before calling myFunction3
   printf("Before calling myFunction3 first_par: %d\n", first_par);
   printf("Before calling myFunction3 second_par: %s\n", second_par);
   printf("Before calling myFunction3 third_par: %d, %d, %d\n\n", third_par[0], third_par[1],
third_par[2]);

   myFunction3(first_par, second_par, &third_par);

   // Values after calling myFunction3
   printf("After calling myFunction3 first_par: %d\n", first_par);
```

```
    printf("After calling myFunction3 second_par: %s\n", second_par);
    printf("After calling myFunction3 third_par: %d, %d, %d\n\n", third_par[0], third_par[1],
third_par[2]);

    // delete from heap
    free(third_par);

    return 0;
}
```

5.2 Output

```
sampleString is the first parameter
7 is the second parameter

Number of paramaters: 1
1

Number of paramaters: 3
1
2
3

Before calling myFunction3 first_par: 7
Before calling myFunction3 second_par: sampleString
Before calling myFunction3 third_par: 1, 2, 3

In myFunction3 par0: 84
In myFunction3 par1: New String
In myFunction3 par2: 4, 5, 6

After calling myFunction3 first_par: 7
After calling myFunction3 second_par: sampleString
After calling myFunction3 third_par: 4, 5, 6
```

5.3 Explanation

In case of the features that I discussed above for the previous languages, C language does not support a few of them. First of all, keyword parameter correspondence is not supported in core of C.

C does not support default parameter values.

In C, to use variable number of parameters, it is needed to include <stdarg.h> library, so that we can use this feature by passing number of parameters and the parameters themselves as arguments to the subprogram like in the example of myFunction2.

C supports only pass-by-value unlike C++, but we can create the ‘illusion of pass-by-reference’ using dereferencing feature of pointers like in the example of myFunction3. However, actually it creates a new memory location with the same address so the value pass in the parameter is affected.

6. Summary

Feature / Language	Positional / Keyword	Default param.	Variable # of param.	Param. Passing
Python	Both	Yes	Yes	Pass-by-object-ref
JavaScript	Both	Yes	Yes	Pass-by-val
PHP	Positional	Yes	Yes	Pass-by-val Pass-by-ref
C++	Positional	Yes	Yes	Pass-by-val Pass-by-ref
C	Positional	No	Yes	Pass-by-val

7. References

- C++ Variadic templates - https://www.geeksforgeeks.org/variadic-function-templates-c/
- Python parameter passing - https://jeffknupp.com/blog/2012/11/13/is-python-callbyvalue-or-callbyreference-neither/
- C++ Keyword parameters - https://softwareengineering.stackexchange.com/questions/329709/python-style-keyword-args-in-c-good-practice-or-bad-idea
- PHP Keyword parameters - https://stackoverflow.com/questions/10181055/php-function-arguments
- JavaScript parameter pass - https://snook.ca/archives/javascript/javascript_pass
- C Variable number of parameters - https://www.tutorialspoint.com/cprogramming/c_variable_arguments.htm
- C Pass-by-Reference - https://stackoverflow.com/questions/2229498/passing-by-reference-in-c