CSE216 – Programming Abstractions
Recitation 4

Objectives:

- Understand parameter passing in Java and Python
- Revise C pointers

Download Recitation4.zip.

Modes of passing parameters:

1. **Pass by value:** Make a copy of the parameter. The most common strategy is the call-by-value evaluation, sometimes also called pass-by-value. This strategy is used in C and C++, for example. In call-by-value, the argument expression is evaluated, and the result of this evaluation is bound to the corresponding variable in the function. So, if the expression is a variable, a local copy of its value will be used, i.e. the variable in the caller's scope will be unchanged when the function returns.

2. **Pass by reference:** Allows the function to change the parameter. In call-by-reference evaluation, which is also known as pass-by-reference, a function gets an implicit reference to the argument, rather than a copy of its value. As a consequence, the function can modify the argument, i.e. the value of the variable in the caller's scope can be changed. The advantage of call-by-reference consists in the advantage of greater time- and space-efficiency, because arguments do not need to be copied. On the other hand this harbours the disadvantage that variables can be "accidentally" changed in a function call. So special care has to be taken to "protect" the values, which shouldn't be changed.

3. **Pass by sharing:** requires parameter to be a reference itself.

    - Makes copy of reference that initially refers to the same object.
    - Within subroutine, value of the object can be changed.
    - However, identity of the object cannot be changed.
    - E.g., User defined Java Objects.

Argument passing in Java

1. Java uses call-by-value for variables of built-in type (all of which are values). See example: PassByValue.java

2. Call-by-sharing for variables of user-defined class types (all of which are references). See example: VehicleProcessor.java

Argument passing in Python[1]

Integer variables: The parameter inside of the function remains a reference to the arguments variable, as long as the parameter is not changed. As soon as a new value will be assigned to it, Python creates a separate local variable.

List variables: List variables behave like integer variables unless they are modified within a function. This results in the side effect of modifying list outside the function. A function is said to have a side effect if, in addition to producing a value, it modifies the caller's environment in other ways. For example, a function might modify a global or static variable, modify one of its arguments, raise an exception, write data to a display or file and so on. The unwanted side effect can be avoided by passing a copy of list to the function.

Tuple variables: Tuple variables are immutable and they cannot be modified within a function. Trying to assign another values to tuple within a function does not have any effect outside the function. If you pass immutable arguments like integers, strings or tuples to a function, the passing acts like call-by-value. The object reference is passed to the function parameters. They can't be changed within the function, because they can't be changed at all, i.e. they are immutable.

Variable length of parameters: The asterisk "*" is used in Python to define a variable number of arguments. The asterisk character has to precede a variable identifier in the parameter list.

See program **parameterpassing.py**.

Exercise: Write a Python program to calculate arithmetic mean of a list or tuple using variable number of arguments.

C Pointers[2]

In C, you can create a special variable that stores the address (rather than the value). This variable is called pointer variable or simply a pointer.

Creating a pointer variable

```
data_type* pointer_variable_name;
```

E.g.: `int* p;`

Above statement defines, p as pointer variable of type int.

[1] Reference: Passing Arguments, https://www.python-course.eu/passing_arguments.php.
[2] Reference: C Pointers, https://www.programiz.com/c-programming/c-pointers.

Reference operator (&) and Dereference operator (*)

& is called reference operator. It gives you the address of a variable. Likewise, there is another operator that gets you the value from the address, it is called a dereference operator *.

Note: The * sign when declaring a pointer is not a dereference operator. It is just a similar notation that creates a pointer.

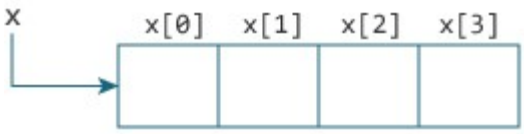
See example: cpointers.c

https://www.programiz.com/c-programming/c-pointers

Relationship between Arrays and Pointers

Consider an array:

int x[4];



x and &x[0] both contains the same address. Hence, &x[0] is equivalent to x. And, x[0] is equivalent to *x. Similarly,

- &x[1] is equivalent to x+1 and x[1] is equivalent to *(x+1).
- &x[2] is equivalent to x+2 and x[2] is equivalent to *(x+2).
- ...
- Basically, &x[i] is equivalent to x+i and x[i] is equivalent to *(x+i).

See example: carraypointers.c

Exercise: Write a C program to calculate arithmetic mean of an array using array pointers.