e-ISBN : 978-629-98755-2-9

SIG : e-Learning@CS
https://appspenang.uitm.edu.my/sigcs/
Publication Date: 18 – Sep - 2024

SEMANTIC MODEL OF PARAMETERS PASSING IN IMPERATIVE PARADIGM WITH PROCEDURAL PROGRAMMING USING C

*Jamal Othman[1], Syarifah Adilah Mohamed Yusoff[2]
**jamalothman@uitm.edu.my[1], syarifah.adilah@uitm.edu.my[2]*

[1,2]Jabatan Sains Komputer & Matematik (JSKM),
Universiti Teknologi MARA Cawangan Pulau Pinang, Malaysia

**Corresponding author*

ABSTRACT

Programming language can be classified as imperative, object-oriented, functional, logic and scripting programming paradigms. Imperative is based on commands that update the variables which exist on the computer memory. Imperative requires functions for every step to solve a problem. Imperative specifies on how the problem is to be solved, which requires a detailed step-by-step instruction. Procedural programming is the derivative of the imperative paradigm which adds functions which are also known as subroutines or procedures. Procedural programming encourages the programmer to subdivide the codes into specific tasks as a function to improve the modularity of the program or looks structured. C programming provides different types of semantic models in terms of parameter passing to the subroutine from the main program or vice versa. The parameters could be variables with primitive data type, arrays or pointers. The type of parameters passing can be characterized into three semantic models such as the in mode, out mode and in-out mode. The semantic models are predetermined through the implementation model either the parameters are passed as pass by value, by pointer or reference, by value-result, by result or by array.

Keywords: imperative, procedural, in mode, out mode, in-out mode

Introduction

Imperative paradigm is the oldest programming approach. The origin of the imperative paradigm is the machine language and assembly language. Imperative programming closest to the actual mechanical behavior of a computer. Imperative program related to sequence of instructions that change the memory state until it achieves the desired end state (Jes´us & Pablo, 2022). The imperative paradigm is useful for small scale applications, but cumbersome for big scale projects and parallel programming. Most of the imperative programming paradigm such as C programming language provides the control structure IF for branch execution and FOR or WHILE for loop execution. GOTO command is also provided for jumping between line executions. Procedural programming is the improvised version of the imperative paradigm which means the execution of the codes have to go through the entire code without skipping any commands. This can be concluded that the GOTO command is not allowed in procedural programming (Bartoníček, 2014). Examples of imperative programming languages are C, FORTRAN, ALGOL and COBOL.

The major strength of the imperative paradigm is its resemblance to the native language of the computer, which makes it efficient to translate and execute the high-level programming language into the imperative paradigm. Procedural programming decreases the expenses of program development as well the system maintenance. Procedural programming reduces duplication of codes or code redundancies. Code duplication is when the program fragment has a similar function in another part of the program fragment. Code duplication complicates the program maintenance or modification since we need to perform similar changes to all duplicated program fragments. Developers are encouraged to reuse the similar code fragment or segment by applying the subroutines or functions (Avacheva & Prutzkow, 2020). Subroutine reduces the length of the codes, increases the system efficiency, cuts the cost of system maintenance, provides modularity and divides the codes into abstraction level (Djikstra, 1968). This article will elaborate three (3) types of parameters passing semantic models either the parameter is passed as in, out or in-out mode.

Parameter Passing Semantic Models

Semantic models of parameter passing in C programming can be classified as in, out and in-out mode. Generally, the in and in-out mode semantic models are applied among practitioners. The following is an example of an in mode parameter passing model in C programming.

```
#include <stdio.h>

void calculateSUM(int,int);//prototype function
int main()
{
int x, y;
printf("\n Enter first number : ");
scanf("%d", &x);
printf("\n Enter second number  : ");
scanf("%d", &y);
calculateSUM(x,y);//calling function
return 0;
}

void calculateSUM(int a, int b) //definition function
{
int sum = a + b;
printf("\n The summation of %d and %d is %d ", a, b, sum);
return;
}
```

Figure 1: C Program with the in-mode parameter passing

The above figure 1, shows that the main program sends two (2) parameters `x` and `y` to the function named `calculateSUM` as pass by value. The function `calculateSUM(…)` receives `x` and `y` from the main program and passes over to `a` and `b` respectively at the function header. Both values of `a` and `b` will be used for arithmetic operations for summation. The result of summation is displayed

and settled in the function. None of the updated values will be sent back to the main program. It shows that the values are sent to the function from the main program and none of the results will be sent back to the main program. This type of parameter passing is also called pass by value. The following diagram named structured chart shows the parameter passing flows between the main program and the subroutine or function.

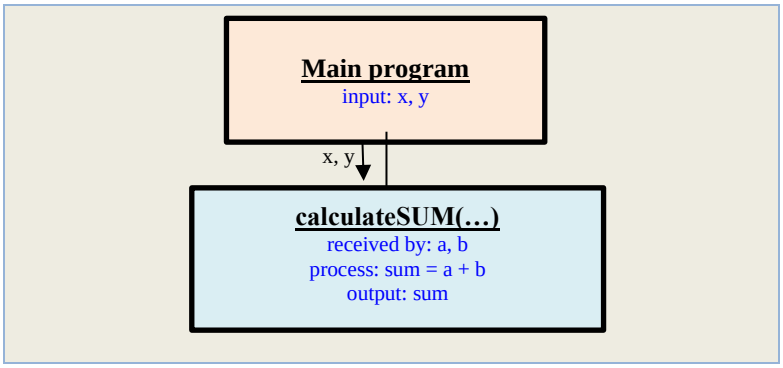


Figure 2: Structured chart that shows the parameter passing flows, in mode semantic model

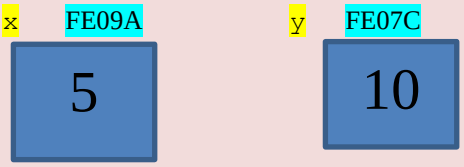
The second type of parameters passing the semantic model is the in-out mode as shown in the following figure 3.

```
#include <stdio.h>

void swap(int*,int*);//prototype function

int main()
{
	int x = 5, y = 10;

	printf ("\n The original value of X is %d ",x);
	printf ("\n The original value of Y is %d ",y);

	swap(&x,&y); //calling function

	printf ("\n The value of X after swap is %d ",x);
	printf ("\n The value of Y after swap is %d ",y);

	return 0;
}

void swap(int* a, int* b) //definition function
{
	int temp;

	temp = *a;
	*a = *b;
	*b = temp;

	return;
}
```

Figure 3: C Program with in-out mode parameter passing

The main function sends two parameters of computer memory address `&x` and `&y` to the `swap(…)` function. The two addresses of `&x` and `&y` will be received by the pointer variables `a` and `b` respectively in the `swap(…)` function header. These two addresses are actually pointed by the pointer variables `a` and `b` indirectly to address `&x` and `&y` respectively. Next, the pointer variable `a` which holds the address of `&x` will be pointing to variable `temp` as temporary. Then, the pointer variable `b` is assigned to pointer variable `a` which means the pointer variable `a` is now pointing to the address of `b`. Later, the variable `temp` is assigned to pointer variable `b`, which means the pointer variable `b` is now pointing to the address of `a`. Finally, the updated address pointed by variable `a` and `b` will be returned back to the main program and received by `x` and `y` with the latest address in the main function. This type of parameter passing is also called pass by value-result. C++ programming language does not provide a sending parameter with the pointer variable as applied in the C programming language (Othman, 2010). Nevertheless, C++ provides the pass by value-result or in-out mode parameter passing semantic model through pass by reference in which none of the pointer variables are involved in the codes. The following figure 4, shows the logical state diagram for the program as shown in figure 3.

```
//int x = 5, y = 10;
```



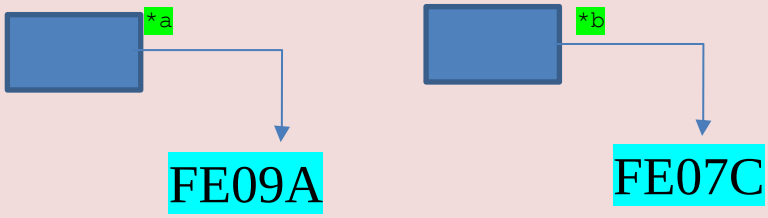
Variable `x` holds the value 5 and is located at the address FE09A.

Variable `y` holds the value 10 and is located at the address FE07C.

```
//swap (%a, &y);
```

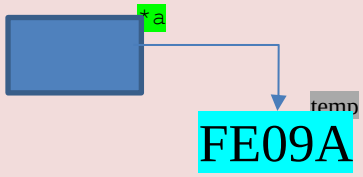
Main program sends FE09A and FE07C to `swap(…)` function through its parameters.

```
//void swap(int* a, int* b)
```



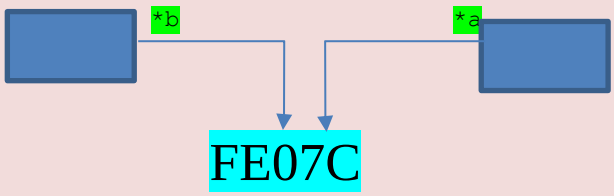
The pointer variable `a is pointing` to the address of `x` and the pointer variable `b` is pointing to the address of `y`.

```
// temp = *a;
```



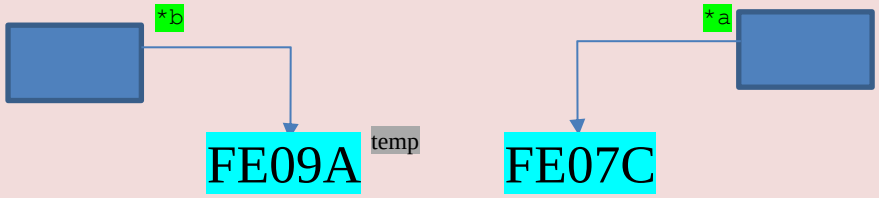
The variable temp has the same similar value of address as pointed by the pointer variable a.

```
//*a = *b;
```



The pointer variables a and b are pointing to the same address.

```
//*b = temp;
```

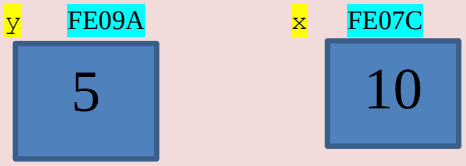


Now, the final state is the pointer variable b is pointing to the address held by the variable temp and the pointer variable a is pointing to the address as previously pointed by the pointer variable b.

Later, when the formal parameters at the header returns the result, it will be returning the exchanged address to the main program.

The main program will be receiving address **FE07C** by the variable x and the address **FE09A** by the variable y. At the location **FE07C**, the memory address holds the value 10 and **FE09A** holds the value 5, which now both variables received the exchanged numbers.

The following is the final memory state of the value held by variable x and y.



Variable x holds the value 10 and is located at the address FE07C.

Variable y holds the value 5 and located at the address FE09A.

Figure 4: Logical state diagram of in-out mode parameter passing semantic model

The third type of the parameter passing semantic model in imperative programming paradigm using C is the out mode as shown in the following figure 5.

```
#include <stdio.h>
int input();//prototype function
int main()
{
	int number = input();//calling function
	printf("\n The number entered in the function is %d",number);
	return 0;
}

int input()//definition function
{
	int x;
	printf("\n Enter a number : ");
	scanf("%d",&x);
	return x;
}
```

Figure 5: C Program with the out-mode parameter passing

The above code as shown in figure 5 illustrates that the input is performed in the function. None of the values are passed from the main program to the subroutine. Once the value is entered in the function, it will be returned to the main program and assigned to a variable for further actions. The out mode parameter passing is also called as pass by result. The following figure 6 shows the structured chart of the parameter passing flows from the subroutine to the main program.

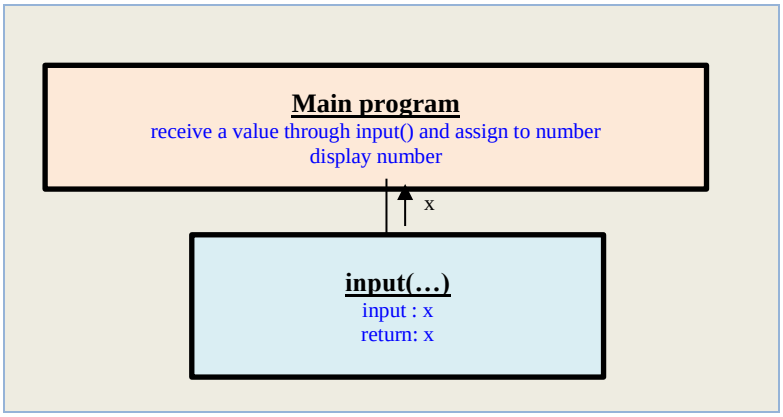


Figure 6: Structured chart that shows the parameter passing flows, out mode semantic model

The three types of parameters passing semantics model as discussed before are the pass by value (in mode), pass by value-result (in-out mode) and pass by result (out mode). The next type of parameter passing is sending the arrays. This type of parameter passing will pass the arrays to the

function, and the affected subscript of an array will be updated and the result will be returned to the main program. The following figure 7 shows the implementation of passing the array as parameter.

```
#include <stdio.h>

void process(int a[], const int S) //definition function
{
	for (int i=0;i<S;i++)
	{
		a[i]= a[i] * 2;
	}
	return;
}

int main()
{
	const int SIZE = 5;
	int x[5]={1,2,3,4,5};

	printf("\n Array contents before processing ");
	for (int i=0;i<SIZE;i++)
	{		printf("%d ",x[i]);		}

	process(x,SIZE); //calling function
	printf("\n Array contents after processing ");
	for (int i=0;i<SIZE;i++)
	{		printf("%d ",x[i]);		}

}
```

Figure 7: C Program with array parameter passing

The contents of array variable `x` before it is sent to the function are `{1,2,3,4,5}`. The function `process(…)` receives the array and multiplies each of the array subscripts by 2 and results `{2,4,6,8,10}`. The output as shown below, depicts that the original contents can be changed if the parameter sent is an array type. Sending an array as part of the parameter to the function is categorized as pass by value-result and the semantic model is the in-out mode.

```
Array contents before processing 1 2 3 4 5
Array contents after processing 2 4 6 8 10
```

Figure 8: Comparison of array contents, before and after processing

Conclusively, the parameter passing types and its semantic model can be summarized as shown in the following table.

Table 1: Parameter passing type and semantic model

Parameter Passing Type	Semantic Model
Pass by value	In mode
Pass by value-result	In-out mode
Pass by Pointer or Reference	In-out mode
Pass by result	Out mode
Pass array	In-out mode

Conclusion

In programming, parameter passing plays a crucial role in passing data between different parts of a program, such as functions or subroutines. The method of passing parameters can significantly impact the efficiency and behavior of a program. By understanding the different methods of parameter passing - including pass-by-value, pass-by-reference and pass-by-pointer, the developers can make informed decisions about which approach to use based on factors such as performance requirements, memory management, and the desired behavior of the program. Each method has its advantages and limitations. Ultimately, the choice of parameter passing method depends on the specific requirements of the program and the trade-offs between performance, memory usage, and data integrity. By carefully considering these factors, developers can design robust and efficient software systems.

References:

Avacheva, T., & Prutzkow, A. (2020), The Evolution of Imperative Programming Paradigms as a Search for New Ways to Reduce Code Duplication, *IOP Conference Series Materials Science and Engineering*, DOI: https://iopscience.iop.org/article/10.1088/1757-899X/714/1/01200

Bartoníček, Jan. (2014), Programming Language Paradigms & The Main Principles of Object-Oriented Programming, *CRIS - Bulletin of the Centre for Research and Interdisciplinary Study*, http://dx.doi.org/10.2478/cris-2014-0006

Jes´us, F., & Pablo, G. (2022), Programming Paradigms: Lectures on High-performance Computing for Economists VII, University Pennsylvania, Retrieved May 8, 2024, from https://www.sas.upenn.edu/~jesusfv/Lecture_HPC_7_Programming_Paradigms.pdf

Djikstra, E. W. (1968), The Structure of the 'THE' – Multiprogramming System, *Communications of the ACM*, vol. 11, number 5, pp. 341-346.

Othman, J. (2010), Fundamentals Of Programming: With Examples in C, C++ and Java, 1st edition, *Pusat Penerbitan Universiti (UPENA), UiTM Malaysia*, ISBN: 978-967-363-110-0.