

Programming Languages
2nd edition
Tucker and Noonan

Chapter 9
Functions

It is better to have 100 functions operate on one data structure than 10 functions on 10 data structures.
A. Perlis

Contents

- 9.1 Basic Terminology
- 9.2 Function Call and Return
- 9.3 Parameters
- 9.4 Parameter Passing Mechanisms
- 9.5 Activation Records
- 9.6 Recursive Functions
- 9.7 Run Time Stack

9.1 Basic Terminology

Value-returning functions:

- known as "non-void functions/methods" in C/C++/Java
- called from within an expression.
e.g., $x = (b*b - \text{sqrt}(4*a*c))/2*a$

Non-value-returning functions:

- known as "procedures" in Ada,
"subroutines" in Fortran,
"void functions/methods" in C/C++/Java
- called from a separate statement.
e.g., `strcpy(s1, s2);`

9.2 Function Call and Return

Example C/C++
Program
Fig 9.1

```
int h, i;
void B(int w) {
    int j, k;
    i = 2*w;
    w = w+1;
}
void A(int x, int y) {
    bool i, j;
    B(h);
}
int main() {
    int a, b;
    h = 5; a = 3; b = 2;
    A(a, b);
}
```

9.3 Parameters

Definitions

- An *argument* is an expression that appears in a function call.
- A *parameter* is an identifier that appears in a function declaration.

E.g., in Figure 9.1

The call `A(a, b)` has arguments `a` and `b`.

The function declaration `A` has parameters `x` and `y`.

Parameter-Argument Matching

Usually by number and by position.

I.e., any call to `A` must have two arguments, and they must match the corresponding parameters' types.

Exceptions:

Perl - parameters aren't declared in a function header.

Instead, parameters are available in an array `@_`, and are accessed using a subscript on this array.

Ada - arguments and parameters can be linked by name. E.g., the call `A(y=>b, x=>a)` is the same as `A(a, b)`

9.4 Parameter Passing Mechanisms

- By value
- By reference
- By value-result
- By result
- By name

Pass by Value

Compute the *value* of the argument at the time of the call and assign that value to the parameter.

E.g., in the call $A(a, b)$ in Fig. 9.1, a and b are passed by value. So the values of parameters x and y become 3 and 2, respectively when the call begins.

So passing by value doesn't normally allow the called function to modify an argument's value.

All arguments in C and Java are passed by value.

But references can be passed to allow argument values to be modified. E.g., `void swap(int *a, int *b) { ... }`

Pass by Reference

Compute the *address* of the argument at the time of the call and assign it to the parameter.

Example
Fig 9.3

Since h is passed by reference, its value changes during the call to B .

```
int h, i;
void B(int* w) {
    int j, k;
    i = 2*( *w);
    *w = *w+1;
}
void A(int* x, int* y) {
    bool i, j;
    B(&h);
}
int main() {
    int a, b;
    h = 5; a = 3; b = 2;
    A(&a, &b);
}
```

Pass by Value-Result and Result

Pass by value at the time of the call and/or copy the result back to the argument at the end of the call.

- E.g., *Ada's in out parameter can be implemented as value-result.*
- *Value-result is often called copy-in-copy-out.*

Reference and value-result are the same, except when *aliasing* occurs. That is, when:

- *the same variable is both passed and globally referenced from the called function, or*
- *the same variable is passed for two different parameters.*

Pass by Name

Textually substitute the argument for every instance of its corresponding parameter in the function body.

- *Originated with Algol 60 (Jensen's device), but was dropped by Algol's successors -- Pascal, Ada, Modula.*
- *Exemplifies **late binding**, since evaluation of the argument is delayed until its occurrence in the function body is actually executed.*
- *Associated with **lazy evaluation** in functional languages*

9.5 Activation Records

A block of information associated with each function call, which includes:

- *parameters and local variables*
- *Return address*
- *Saved registers*
- *Temporary variables*
- *Return value*
- *Static link - to the function's static parent*
- *Dynamic link - to the activation record of the caller*

9.6 Recursive Functions

A function that can call itself, either directly or indirectly, is a recursive function. E.g.,

```
int factorial (int n) {
    if (n < 2)
        return 1;
    else return n*factorial(n-1);
}
```

self-call

9.7 Run Time Stack

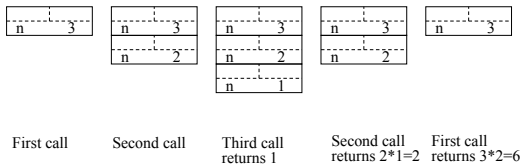
A stack of activation records.

- Each new call pushes an activation record, and each completing call pops the topmost one.
- So, the topmost record is the most recent call, and the stack has all active calls at any run-time moment.

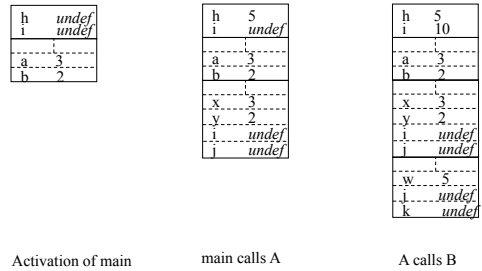
For example, consider the call **factorial(3)**.

This places one activation record onto the stack and generates a second call **factorial(2)**. This call generates the call **factorial(1)**, so that the stack gains three activation records.

Stack Activity for the Call factorial(3) Fig. 9.7



Stack Activity for Program in Fig. 9.1 Fig. 9.8 (links not shown)



Additional Example

http://www.enel.ucalgary.ca/People/Norman/engg335_fall1998/activ_rec/#intmainvoid