Process abstractions are represented in programming languages by subprograms. A subprogram definition describes the actions represented by the subprogram. A subprogram call enacts those actions.

Formal parameters are the names that subprograms use to refer to the actual parameters given in subprogram calls.

Subprograms can be either functions, which model mathematical functions and are used to define new operations, or procedures, which define new statements.

Local variables in subprograms can be stack dynamic, providing support for recursion, or static, providing efficiency and history-sensitive local variables.

There are three fundamental semantics models of parameter passing—indirect mode, out mode, and inout mode—and a number of approaches to implementation.

Aliasing can occur when pass-by-reference parameters are used, both among two or more parameters and between a parameter and an accessible nonlocal variable.

Parameters that are subprogram names provide a necessary service but are sometimes difficult to understand. The opacity lies in the referencing environment that is available when a subprogram that has been passed as a parameter is executing.

Ada, C++, and C# allow both subprogram and operator overloading. Subprograms can be overloaded as long as the various versions can be disambiguated by the types of their parameters or returned values. Function definitions can be used to build additional meanings for operators.

Subprograms in Ada and C++ can be generic, using parametric polymorphism, so the desired types of their data objects can be passed to the compiler, which then can construct units for the requested types.

A coroutine is a special subprogram that has multiple entries. They can be used to provide interleaved execution of subprograms.

REVIEW QUESTIONS

1. What are the three general characteristics of subprograms?
2. What does it mean for a subprogram to be active?
3. What is a parameter profile? What is a subprogram protocol?
4. What are formal parameters? What are actual parameters?
5. What are the advantages and disadvantages of keyword parameters?
6. What are the design issues for subprograms?
7. What are the advantages and disadvantages of dynamic local variables?
8. What are the three semantic models of parameter passing?
9. What are the modes, the conceptual models of transfer, the advantages, and the disadvantages of pass-by-value, pass-by-result, pass-by-value-result, and pass-by-reference parameter-passing methods?
10. In what ways can aliases occur with pass-by-reference parameters?
11. What is the difference between the way original C and C89 deal with an actual parameter whose type is not identical to that of the corresponding formal parameter?
12. What is the problem with Ada's policy of allowing implementors to decide which parameters to pass by reference and which to pass by value-result?
13. What are two fundamental design considerations for parameter-passing methods?
14. What are the two issues that arise when subprogram names are parameters?
15. Define shallow and deep binding for referencing environments of subprograms that have been passed as parameters.
16. What is an overloaded subprogram?
17. What is parametric polymorphism?
18. What causes a C++ template function to be instantiated?
19. What are the design issues for functions?
20. In what ways are coroutines different from conventional subprograms?

---

**Problem Set**

1. What are arguments for and against a user program building additional definitions for existing operators, as can be done in Ada and C++? Do you believe such user-defined operator overloading is good or bad? Support your answer.
2. In most Fortran IV implementations, parameters were passed by reference, using access path transmission only. State both the advantages and disadvantages of this design choice.
3. Argue in support of the Ada 83 designers' decision to allow the implementor to choose between implementing in-out mode parameters by copy or by reference.
4. Suppose you wish to write a method that prints a heading on a new output page, along with a page number that is 1 in the first activation and that increases by 1 with each subsequent activation. Can this be done without parameters and without reference to nonlocal variables in Java?

---

**Programming Exercises**

5. Consider the following program written in C syntax:

```c
void main () {
  int value = 2, list[5] = {1, 3, 5, 7, 9};
  swap (value, list[0]);
  swap (list[0], list[1]);
  swap (value, list[value]);
}

void swap (int a, int b) {
  int temp;
  temp = a;
  a = b;
  b = temp;
}
```

For each of the following parameter-passing methods, what are all of the values of the variables `value` and `list` after each of the three calls to `swap`?

a. Passed by value
b. Passed by reference
c. Passed by value-result

6. Present one argument against providing both static and dynamic local variables in subprograms.

7. Argue against the C design of providing only function subprograms.

8. From a textbook on Fortran, learn the syntax and semantics of statement functions. Justify their existence in Fortran.

9. Study the methods of user-defined operator overloading in C++ and Ada, and write a report comparing the two using our criteria for evaluating languages.

10. C# supports out-mode parameters, but neither Java nor C++ does. Give an explanation of this difference.
3. Write a C# or Ada program that determines when the address of an out-mode parameter is computed (at the time of the call or at the time execution of the subprogram finishes).

4. Write a Perl program that passes by reference a literal to a subprogram, which attempts to change the parameter. Given the overall design philosophy of Perl, explain the results.

5. Repeat Exercise 4 in C#.

6. Write a program in some language that has both static and stack-dynamic local variables in subprograms. Create six large (at least 100 x 100) matrices in the subprogram, three static and three stack dynamic. Fill two of the static matrices and two of the stack-dynamic matrices with random numbers in the range of 1 to 100. The code in the subprogram must perform a large number of matrix multiplication operations on the static matrices and time the process. Then it must repeat this with the stack-dynamic matrices. Compare and explain the results.