Chapter 8: Subprograms

Principles of Programming Languages

Contents

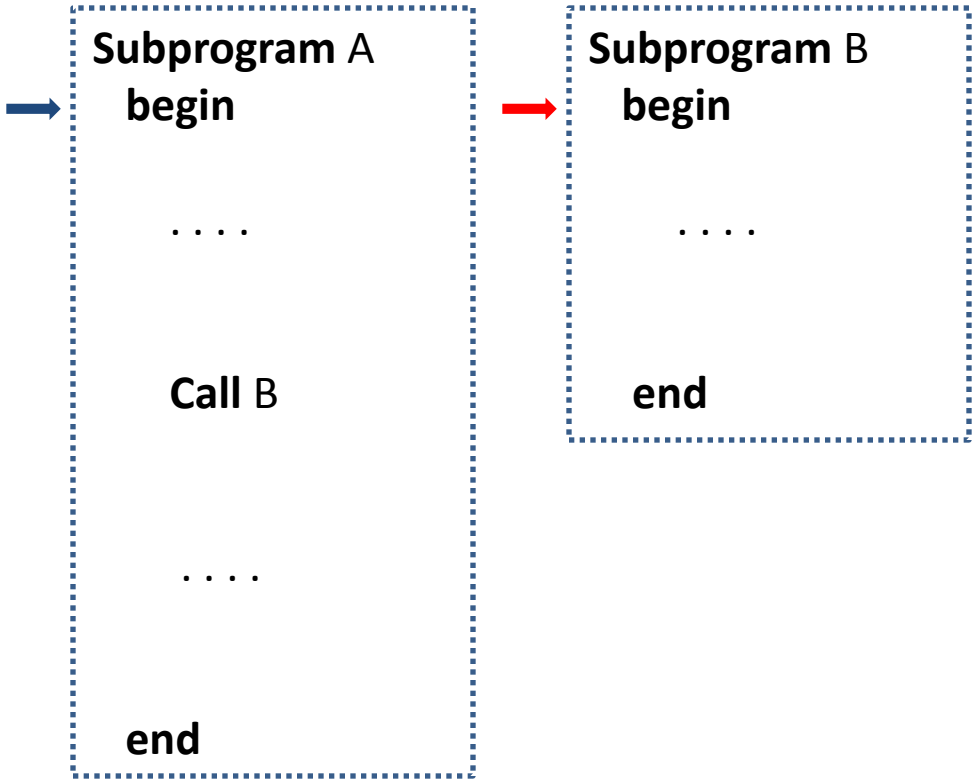
- Fundamentals of Subprograms
- Parameter-Passing Methods
- Overloaded Subprograms
- Generic Subprograms
- Functions
- User-Defined Overloaded Operators
- Coroutines

Introduction

- Two fundamental abstraction facilities
  - Process abstraction
    - Reuse a collection of statements
    - Abstracting the details of a computation by calling subprogram's name
  - Data abstraction
- How about methods of object-oriented languages?

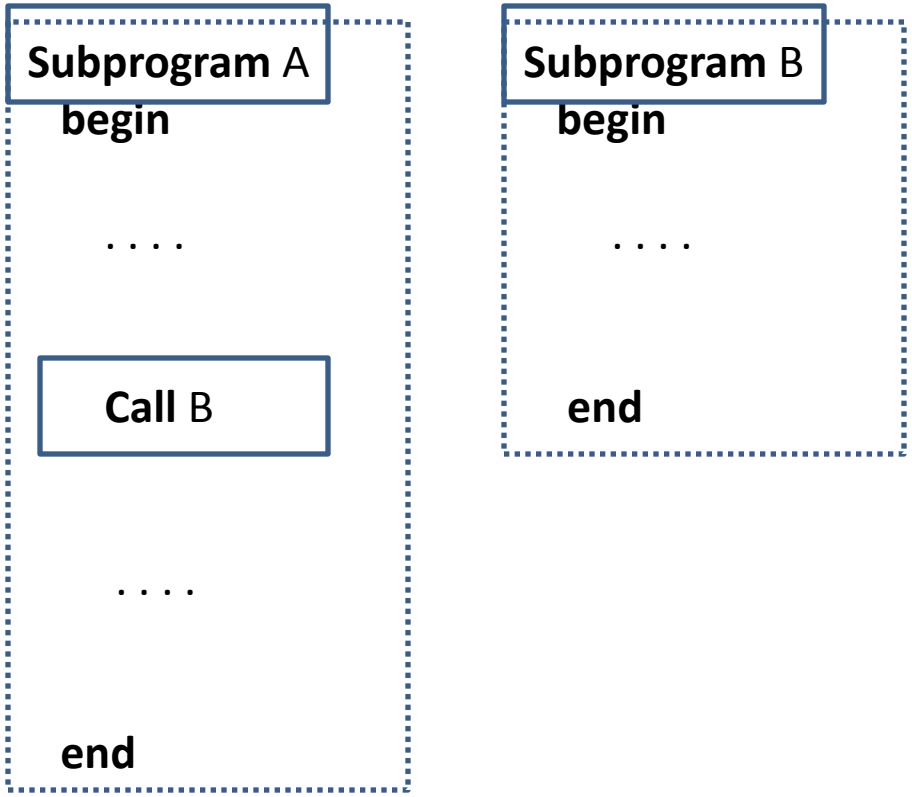
General Characteristics

- Each subprogram has a single entry point
- There is only one subprogram in execution at any given time
- Control always returns to the caller when the called subprogram's execution terminates



Basic Definitions

- Subprogram definition
- Subprogram call
- Subprogram header



Header Examples

```
Subroutine Adder(parameters)
```
Fortran

```
procedure Adder(parameters)
```
Ada

```
def adder(parameters)=
```
Scala

```
void adder(parameters)
```
C

- Specify a kind of subprogram
- Subprogram name
- Optionally a list of parameters

Special Case: C/C++

- Parameter profile
- Protocol
- A *subprogram declaration* provides the protocol, but not the body, of the subprogram
- Function declarations in C and C++ are often called *prototypes*
- **Reason**: not allow forward references to subprogram

Parameters

- Formal parameters vs. actual parameters
- Positional (*all proglang*)
  - The binding of actual parameters to formal parameters is by position: the first actual parameter is bound to the first formal parameter and so forth
  - Safe and effective
- Keyword (*Python, Ada, Fortran 95, Python*)
  - The name of the formal parameter to which an actual parameter is to be bound is specified with the actual parameter
  - Parameters can appear in any order

Parameters

- In certain languages, formal parameters can have default values

  Python:

```
def compute_pay(income, exemptions = 1, tax_rate)
pay = compute_pay(20000.0, tax_rate = 0.15)
```

  C++:

```
float compute_pay(float income, float tax_rate,
                  int exemptions = 1)
```

- C#: accept a variable number of parameters as long as they are of the same type

```
public void Display(params int[] list)
```

Procedures and Functions

- Procedures are collection of statements
  - Produce results by changing non-local variables or formal parameters that allow the transfer of data to the caller
- Functions structurally resemble procedures but are semantically modeled on mathematical functions
  - They are expected to return a value and produce no side effects

Design Issues for Subprograms

- Are local variables static or dynamic?
- Can subprogram definitions appear in other subprogram definitions?
- What parameter-passing methods are used?
- Are parameter types checked?
- Can subprograms be overloaded?
- Can subprogram be generic?

Local Variables

- Stack-dynamic
  - Where does the name come from?
  - Advantages
    - Support for recursion
    - Storage is shared among some subprograms
  - Disadvantages
    - Cost of allocation/de-allocation, initialization
    - Indirect addressing
    - Subprograms cannot be history sensitive
  - Default in most contemporary languages

Local Variables

- Local variables can be static
  - Storage?
  - Advantages
    - Slightly more efficient: no indirection, no run-time overhead
    - Allow subprograms to be history-sensitive
  - Disadvantages:
    - Cannot support recursion
    - Storage cannot be shared
  - Supported in C/C++, Fortran 95

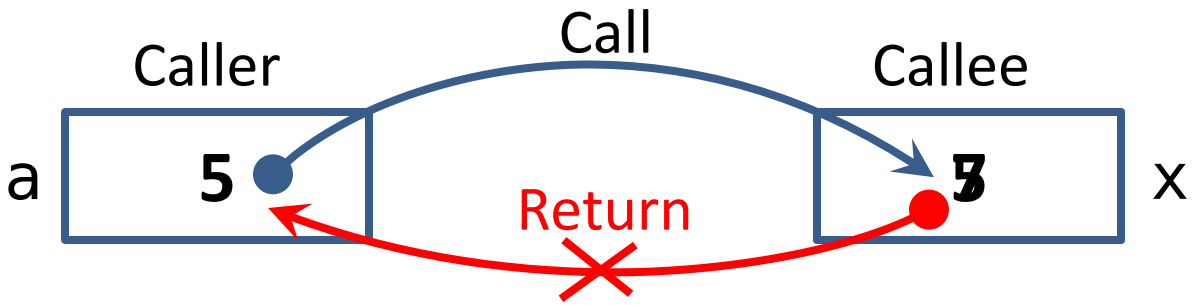
Nested Subprograms

- Originating from Algol 60: Algol 68, Pascal, Ada. Recently, JavaScript, Python, and Ruby
- Hierarchy of both logic and scopes
- Usually with static scoping: grant access to nonlocal variables in enclosing subprograms
- Problems: Chapter 5
- Examples: many in your Tutorial on Scopes with Ada

Parameter-Passing Methods

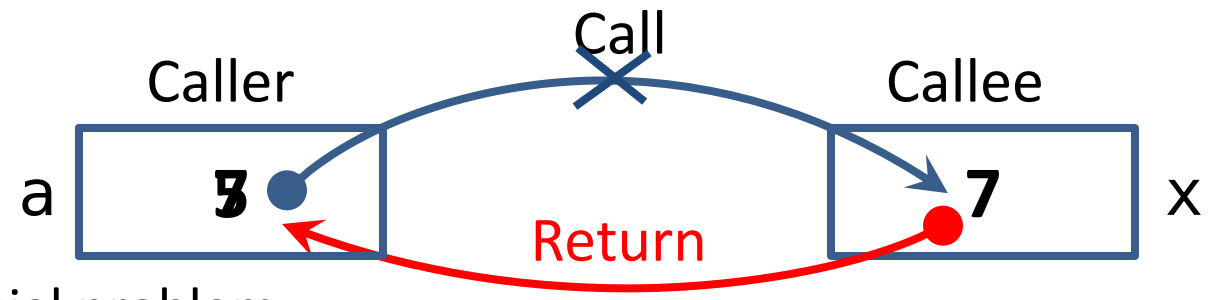
- Ways in which parameters are transmitted to and/or from called subprograms
- Semantics models: formal parameters can
  - Receive data from actual parameters: **in mode**
  - Transmit data to actual parameters: **out mode**
  - Do both: **inout mode**
- Data transfer models:
  - Actual value is copied
  - Access path is transmitted

Pass-by-Value (In Mode)



- Normally implemented by copying, or
- Transmitting an access path but have to **enforce write protection**
- Advantage: fast for scalars
- Disadvantages:
  - When copies are used, additional storage is required
  - Storage and copy operations can be costly

Pass-by-Result (Out Mode)

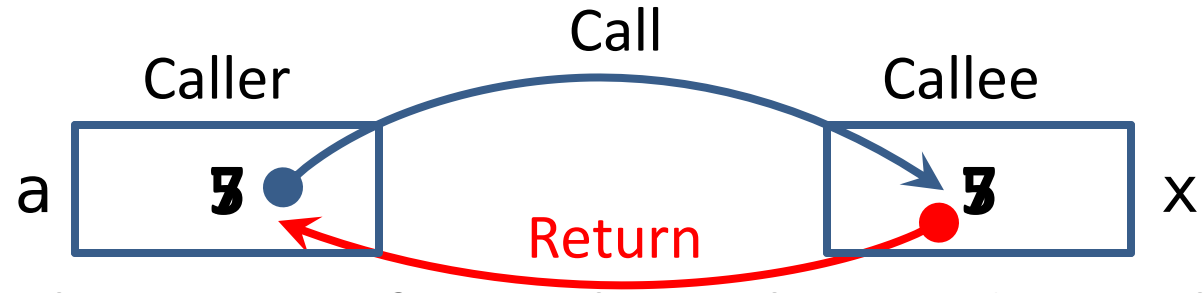


- Potential problem:

```
void Fixer(out int x, out int y) {                    C#
   x = 17; y = 35;
}
f.Fixer(out a, out a);
```

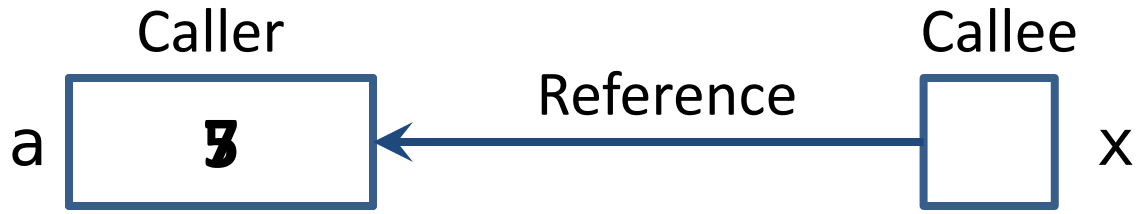
```
void DoIt(out int x, out int index) {                 C#
   x = 17; index = 42;
}
sub = 21;
f.DoIt(out list[sub], out sub)
```

Pass-by-Value-Result (Inout Mode)



- A combination of pass-by-value and pass-by-result
- Sometimes called pass-by-copy
- Formal parameters have local storage
- Disadvantages:
  - Those of pass-by-result
  - Those of pass-by-value

Pass-by-Reference (Inout Mode)



- Pass an access path, usually just an address
- Passing process is efficient (no copying and no duplicated storage)
- Disadvantages
  - Slower accesses (compared to pass-by-value) to formal parameters
  - Potentials for un-wanted side effects
  - Un-wanted aliases (access broadened)

Pass-by-Name (Inout Mode)

- By textual substitution
- Actual binding to a value or address takes place at the time of a reference or assignment

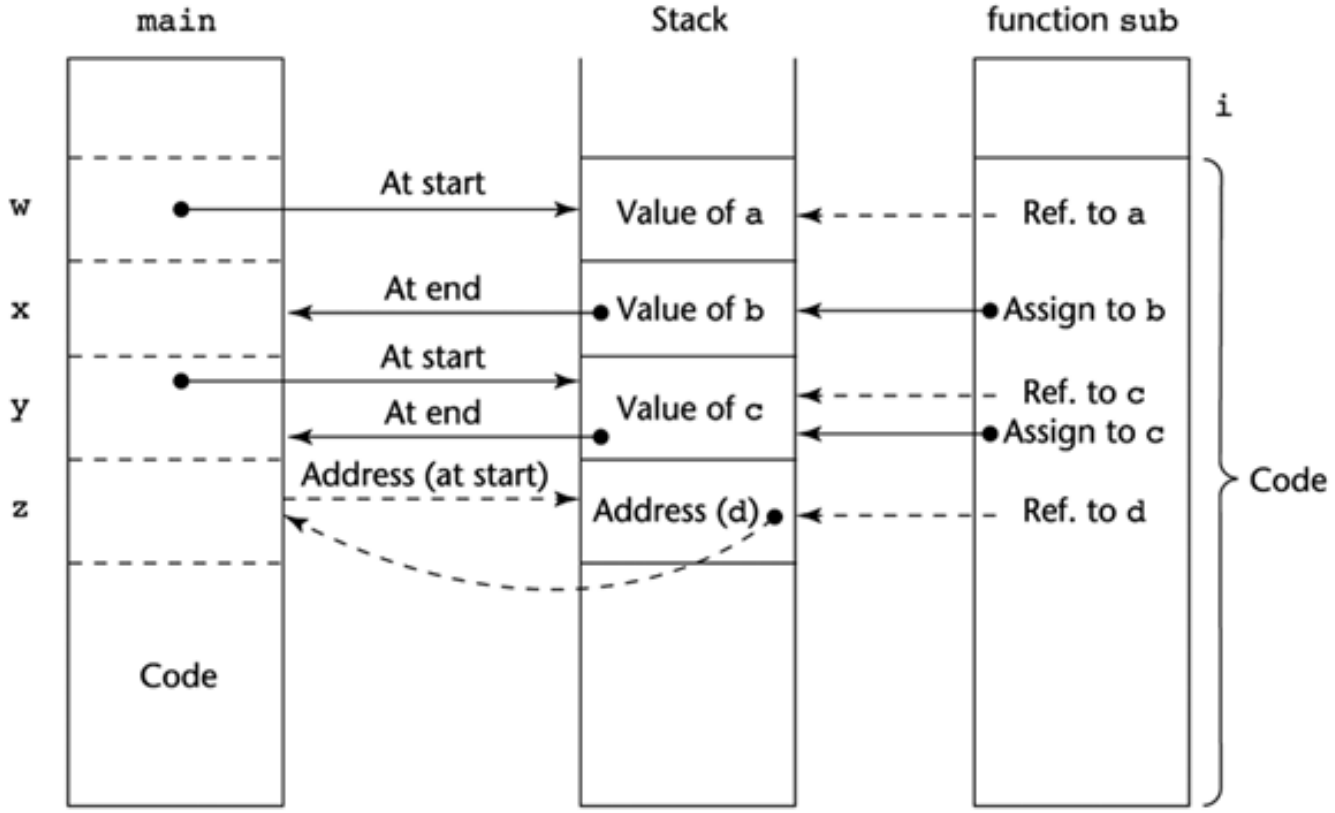
Pass-by-Name (Inout mode)

```
type VECT = array [1..3] of integer;
procedure SUB2 (name I, J: integer);
begin
   I := I + 1;
   J := J + 1;
   write(I, J)
end;
procedure SUB1;
var A: VECT;
    K: integer;
begin
   A[1] := 7; A[2] := 8; A[3] := 9;
   K := 2;
   SUB2(K, A[K]);
   for K := 1 to 3 do write(A[K])
end;
```

```
K := K + 1;
A[K] := A[K] + 1;
write(K,A[K])
```

Implementing Parameter-Passing Methods

- Parameter communication often takes place through the run-time stack



Parameter-Passing Methods in Common Language

- C
  - Default: pass-by-value
  - Pass-by-reference is achieved by using pointers as parameters
- C++
  - Using reference type for pass-by-reference
  - How about constant reference?
- Java
  - All parameters are passed by value
  - Objects copy its reference

Parameter-Passing Methods in Common Language

- Ada
  - Three semantics modes of parameter transmission: **in**, **out**, **in out**; in is the default mode
- C#
  - Default method: pass-by-value
  - Pass-by-reference: **ref**
  - Out mode: **out**

Type Checking Parameters

- Considered very important for reliability
- Original C: no type checking
- C++: yes, but can pass by using ellipsis. For example, `printf` function
- C#: coercions in pass-by-value, no coercions in pass-by-reference
- Python, Ruby: formal parameters are typeless, no type checking is needed

Multidimensional Arrays as Parameters

- If a multidimensional array is passed to a subprogram and the subprogram is separately compiled, the compiler needs to know the declared size of that array to build the storage mapping function
- C/C++:
  - must declare the matrix size in parameters
  - macro is a good choice
- Java, C#: every matrix has length function

Design Considerations

- Two important considerations
  - Efficiency
  - One-way or two-way data transfer is needed
- But the above considerations are in conflict
  - Good programming suggest limited access to variables, which means one-way whenever possible
  - But pass-by-reference is more efficient to pass structures of significant size

Parameters That Are Subprograms

- It is sometimes convenient to pass subprogram names as parameters
- Issues:
  - Are parameter of transferring subprogram type-checked?
  - In languages that allow nested subprogram, what referencing environment for executing the passed subprogram should be used?

Parameters as Subprograms: Type Checking

- *Are parameter of transferring subprogram type-checked?*
- C and C++: functions cannot be passed as parameters but pointers to functions can be passed; parameters can be type checked
- Ada does not allow subprogram parameters; a similar alternative is provided via Ada's generic facility

Parameters as Subprograms: Referencing Environment

- *What referencing environment for executing the passed subprogram should be used?*
- Shallow binding: The environment of the call statement that enacts the passed subprogram
- Deep binding: The environment of the definition of the passed subprogram
- Ad hoc binding: The environment of the call statement that passed the subprogram

Parameters as Subprograms: Referencing Environment

```
function sub1() {
   var x;
   function sub2() {
      alert(x);
   };
   function sub3() {
      var x;
      x = 3;
      sub4(sub2);
   };
   function sub4(subx) {
      var x;
      x = 4;
      subx();
   };
   x = 1;
   sub3();
}
```

- Shallow binding:

  4

- Deep binding:

  1

- Ad hoc binding:

  3

Overloaded Subprograms

- An overloaded subprogram is one that has the same name as another subprogram in the same referencing environment
  - Every version of an overloaded subprogram has a unique protocol (number, order, types of params and return type)
- C++, Java, C#, and Ada include predefined overloaded subprograms
- In Ada, the return type of an overloaded function can be used to disambiguate calls (thus two overloaded functions can have the same parameters)

Generic Subprograms

- A polymorphic subprogram takes parameters of different types on different activations
- Overloaded subprograms: ad hoc polymorphism
- Generic subprograms: parametric polymorphism
- Ada, C++, Java 5.0, C# 2005 (Scala)

Examples of parametric polymorphism: C++

```
template <class Type>
Type max(Type first, Type second) {
    return first > second ? first : second;
}
```

- The above template can be instantiated for any type for which operator > is defined

```
int max (int first, int second) {
     return first > second? first : second;
}
```

Design Issues for Functions

- Are side effects allowed?
  - Parameters should always be in-mode to reduce side effect (like Ada)
- What types of return values are allowed?
  - Most imperative languages restrict the return types
  - C allows any type except arrays and functions
  - C++ is like C but also allows user-defined types
  - Python, Ruby: functions are first class object
  - Java and C# do not have functions but methods can return any type, except method (not a type)

Design Issues for Functions

- Number of returned values?
  - In most languages, only a single value can be returned from a function
  - Ruby: return an array of value if there are more than one expression

User-Defined Overloaded Operators

- Operators can be overloaded in Ada, C++, Python and Ruby
- Data Structures and Algorithms

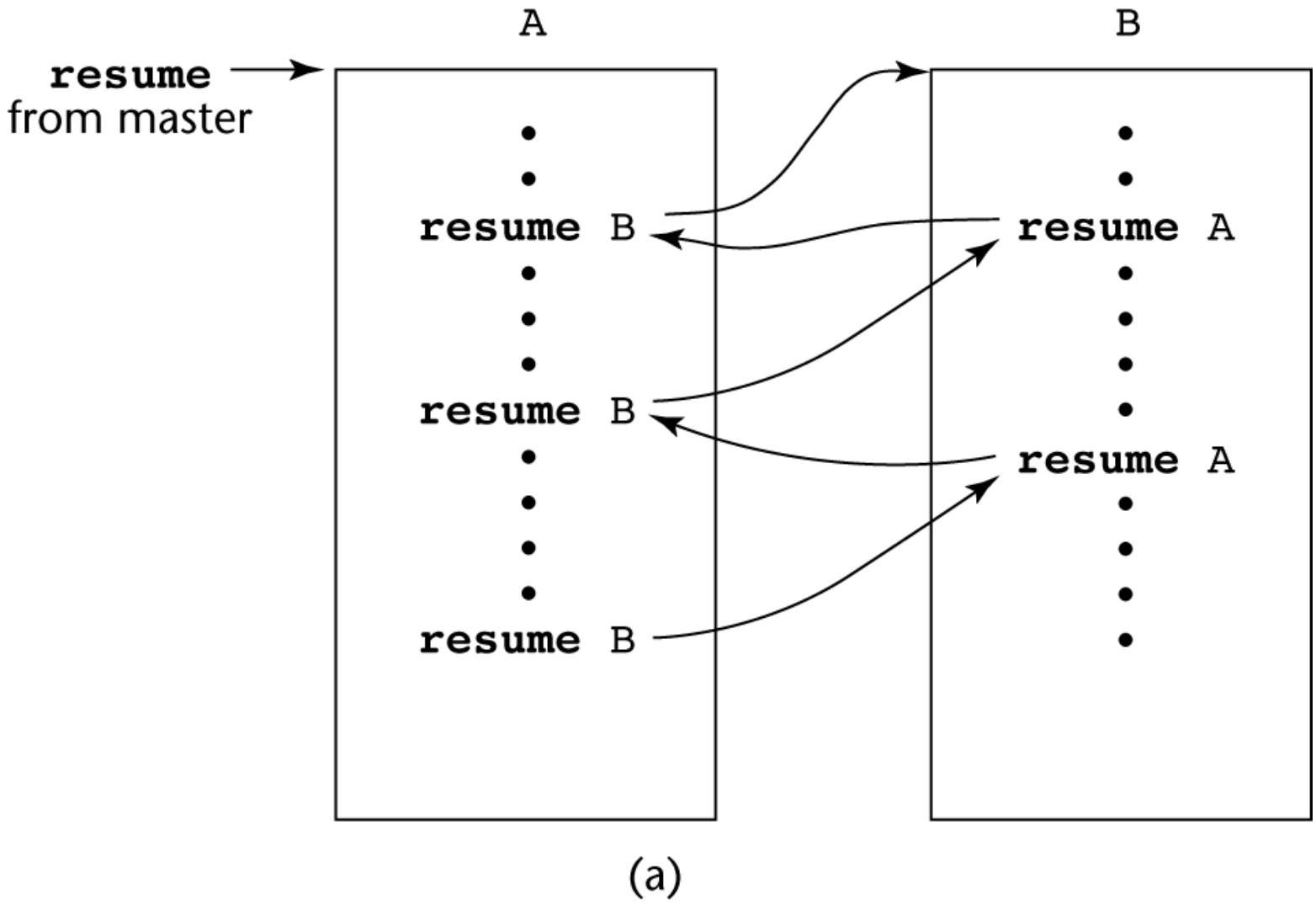
```
int operator *(const vector &a, const vector &b, int len);
```

Coroutines

- A *coroutine* is a subprogram that has multiple entries and controls them itself
- A coroutine call is named a *resume*

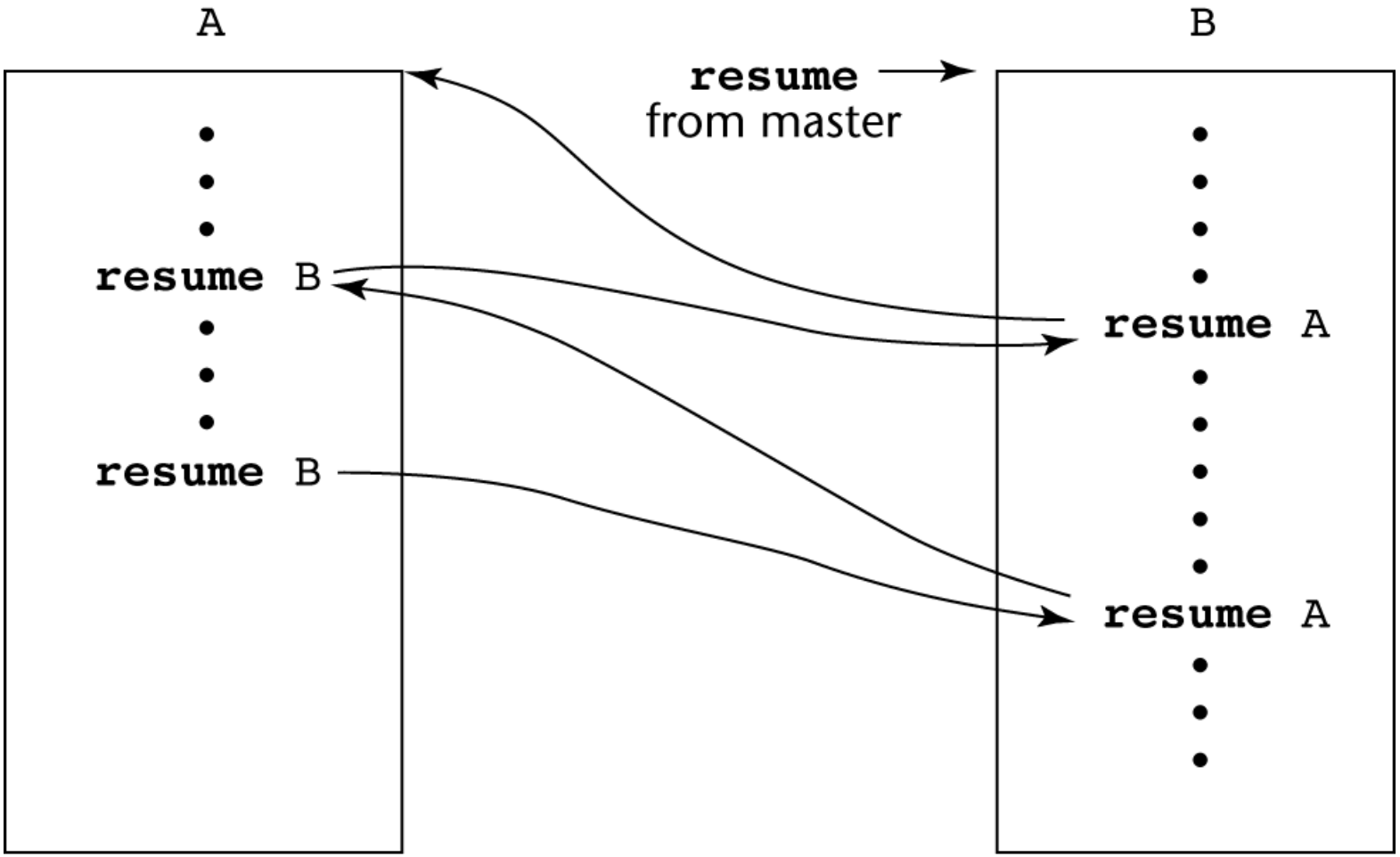
```
sub co1() {
   . . .
   resume co2();
   . . .
   resume co3();
   . . .
}
```

Coroutines Illustrated: Possible Execution Controls



(a)

Coroutines Illustrated: Possible Execution Controls



(b)

Coroutines

- Coroutines repeatedly resume each other
- Coroutines provide quasi-concurrent execution of program units (the coroutines); their execution is interleaved, but not overlapped
- Coroutines are created by a program unit called the master unit

Coroutines Illustrated

- Simulation of a card game
- Four players will have four coroutines, each with collection, or hand, of cards
- Master program then start by resuming one of the player coroutines
- After this player played its turn, could resume the next player coroutine and so forth

Summary

- A subprogram definition describes the actions represented by the subprogram
- Subprograms can be either functions or procedures
- Local variables in subprograms can be stack-dynamic or static
- Three models of parameter passing: in mode, out mode, and inout mode
- Some languages allow operator overloading
- Subprograms can be generic
- A coroutine is a special subprogram with multiple entries