Alexander J King

Comparative Programming Languages

Homework #9

4.6.20

1. Explain each of the following (in your own words:

a. The 3 fundamental rules of subprograms

Each subprogram has a single entry point.

The calling program is suspended during the execution of the called subprogram.

Control always returns to the caller when the called subprograms execution terminates.

b. Subprogram A program within a program that is called by the main program.

c. Subprogram call a request that the subprogram execute.

d. Subprogram Header The kind of subprogram and the parameters.

e. Parameter profile/Protocol Parameter profile and or return type.

f. Formal Parameter –v- Actual Parameter – A formal parameter is a dummy variable listed in the subprogram header and used in the subprogram while an actual parameter represents a value or address used in the subprogram call statement.

g. Procedure –v- Function Procedures are collections of statements that define parameterized computations while functions structurally resemble procedures but are semantically modeled on mathematical functions.

h. Design issues for subprograms Are local variables static or dynamic? Can subprogram definitions appear in other subprogram definitions? What parameter passing methods are provided? Are parameter types checked? If subprograms can be passed as parameters and subprograms can be nested, what is the referencing environment of a passed subprogram? Can subprograms be overloaded? Can subprogram be generic. If the language allows nested subprograms, are

closures supported?

I. Stack dynamic –v- static variables – Stack-dynamic variblable have support for recurion and storage for locals is shared among some subprograms but allocation and de-allocation, initialization time takes longer. Dynamic also uses indirect addressing and subprograms cannot be history sensitive.

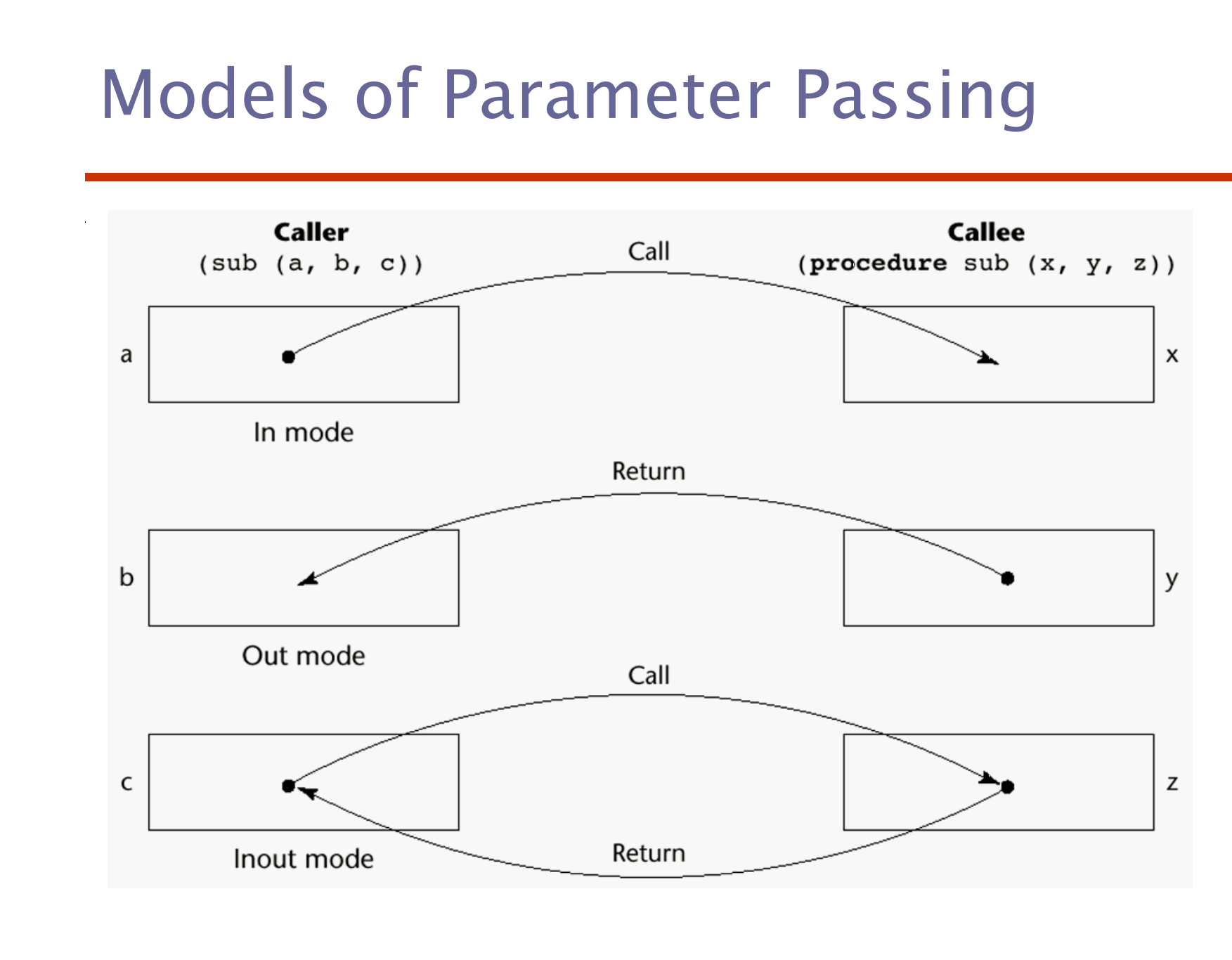
Local static variables are the opposite of stack-dynamic.

j. Parameter passing - In mode Out mode Inout mode

i. In-mode

ii. Out-mode

iii. InOut mode



1. Pass by result (out 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

2. Pass by reference - Pass an access path

• Also called pass-by-sharing

• Advantage: Passing process is efficient (no

copying and no duplicated storage)

• Disadvantages

– Slower accesses (compared to pass-by-value) to

formal parameters

– Potentials for unwanted side effects (collisions)

– Unwanted aliases (access broadened)

3. Pass by name - By textual substitution

• Formals are bound to an access method at

the time of the call, but actual binding to a

value or address takes place at the time of

a reference or assignment

• Allows flexibility in late binding

• Implementation requires that the

referencing environment of the caller is

passed with the parameter, so the actual

parameter address can be calculated

k. Design issues for parameter passing Are local variables static or dynamic?

• Can subprogram definitions appear in other

subprogram definitions?

• What parameter passing methods are provided?

• Are parameter types checked?

• If subprograms can be passed as parameters and

subprograms can be nested, what is the

referencing environment of a passed subprogram?

• Can subprograms be overloaded?

• Can subprogram be generic?

• If the language allows nested subprograms, are

closures supported?

l. Overloaded subprogram -   
 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

• 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)

• Ada, Java, C++, and C# allow users to write

multiple versions of subprograms with the same

name

m. Generic subprogram - A generic or polymorphic subprogram takes

parameters of different types on different

activations

• Overloaded subprograms provide ad hoc

polymorphism

• Subtype polymorphism means that a variable of

type T can access any object of type T or any type

derived from T (OOP languages)

• A subprogram that takes a generic parameter that

is used in a type expression that describes the type

of the parameters of the subprogram provides

parametric polymorphism

- A cheap compile-time substitute for dynamic

binding

n. 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

– Ada subprograms can return any type (but Ada

subprograms are not types, so they cannot be returned)

– Java and C# methods can return any type (but because

methods are not types, they cannot be returned)

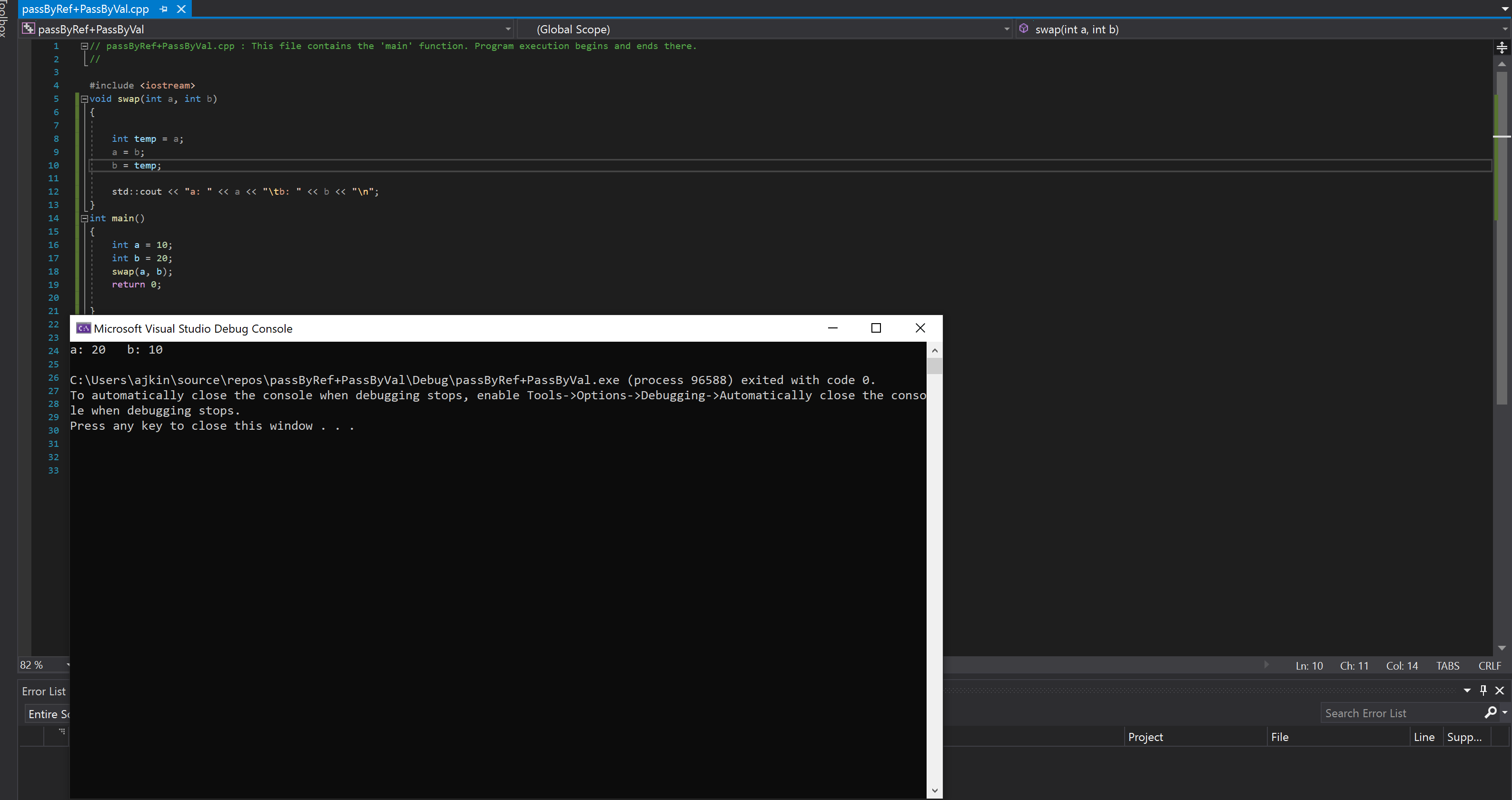
– Python and Ruby treat methods as first-class objects, so

they can be returned, as well as any other class

– Lua allows functions to return multiple values

2. Create a C++ program which calls a subprogram in which the pass-by reference and pass-by value result of at least one parameter produces different results.

Swap by value:



Swap by reference: (Only difference in void swap (int &a, int &b)

