Why dynamic scope?

(Slightly) more modern instances of dynamic scope:
The two ways of thinking correspond closely to the two standard
value).

existing object (or makes it undefined if the definition provides no

variable "a" local to procedure second is no longer active.

This is the most recent, active binding when procedure "first" is
run time we create a binding for "a" when we enter the main program.

require that we choose the most recent, active binding at run time.  At
namely the global variable "a".  Dynamic scope rules, on the other hand,
that the reference resolve to the closest surrounding compile
"first" changes the variable "a" declared in the main program or the

At issue is whether the assignment to the variable "a" in procedure
elaborated at run time.

To resolve a reference, we use the most recent, active binding
physical (lexical) structure of the program.

Dynamic Scope

Declarations that are

let rec...  

let rec

--------

Modula

--------

C++ and Java dispense with declare

--------

Ada, C, C++, Java, and others say scope extends from a declaration to
end of the current block.

Note that there can be arbitrary distance between the (illegal) use
Clearly the programmer meant to use the outer

Or this in Pascal:
Language rules say the second declaration of A covers all of
The first declaration of A causes the static variable to be initialized
with the initializer on that declaration.  A static variable declared in

Two

Declaration Order

----------------------------------------

Dynamic scope rules are elaborated at run time.  They cannot always be resolved by examining the source code.

Dynamic Scope

The two ways of thinking differ at least in that bindings are defined by
the history of program execution, not just by the statements in the
program.

When dynamic scope rules dominate the history of argument
passing, one can explicitly choose to pass the argument to the

Passes down the call stack,


import


C, Ada, C++, Java, and others say scope extends from a declaration to

the end of the current block.

Note that there can be arbitrary distance between the (illegal) use
Clearly the programmer meant to use the outer

Or this in Pascal:
Language rules say the second declaration of A covers all of
The first declaration of A causes the static variable to be initialized
with the initializer on that declaration.  A static variable declared in

Two

Declaration Order

----------------------------------------

Dynamic scope rules are elaborated at run time.  They cannot always be resolved by examining the source code.

Dynamic Scope

The two ways of thinking differ at least in that bindings are defined by
the history of program execution, not just by the statements in the
program.

When dynamic scope rules dominate the history of argument
passing, one can explicitly choose to pass the argument to the

Passes down the call stack,


import


C, Ada, C++, Java, and others say scope extends from a declaration to

the end of the current block.

Note that there can be arbitrary distance between the (illegal) use
Clearly the programmer meant to use the outer

Or this in Pascal:
Language rules say the second declaration of A covers all of
The first declaration of A causes the static variable to be initialized
with the initializer on that declaration.  A static variable declared in

Two

Declaration Order

----------------------------------------

Dynamic scope rules are elaborated at run time.  They cannot always be resolved by examining the source code.

Dynamic Scope

The two ways of thinking differ at least in that bindings are defined by
the history of program execution, not just by the statements in the
program.

When dynamic scope rules dominate the history of argument
passing, one can explicitly choose to pass the argument to the

Passes down the call stack,


import


C, Ada, C++, Java, and others say scope extends from a declaration to

the end of the current block.

Note that there can be arbitrary distance between the (illegal) use
Clearly the programmer meant to use the outer

Or this in Pascal:
Language rules say the second declaration of A covers all of
The first declaration of A causes the static variable to be initialized
with the initializer on that declaration.  A static variable declared in

Two

Declaration Order

----------------------------------------

Dynamic scope rules are elaborated at run time.  They cannot always be resolved by examining the source code.

Dynamic Scope

The two ways of thinking differ at least in that bindings are defined by
the history of program execution, not just by the statements in the
program.

When dynamic scope rules dominate the history of argument
passing, one can explicitly choose to pass the argument to the

Passes down the call stack,