Patrick Austin  
CS 326

Homework # 3

10/6/2015

Patrick Austin

CS 326 Homework 3

1.

a. Compile time. C++ uses static scoping, so scoping must be determined at compile time and cannot be determined at run time, as would happen with a language that uses dynamic scoping.

b. Language implementation time. This is when an implementation decision about the number of bits of a fundamental type would be decided, which would dictate the range of possible values for an integer.

c. Language design time. This is when the nature of the fundamental types for the language would be decided.

d. Run time. Since local variables are dynamically allocated on the stack, the address of a local variable will be bound when a subroutine is entered during run time.

e. Link time. The linker is what resolves references to library functions in the other modules of the program, and it will bind the address to a needed library function.

f. Compile time. C++ does not have nested subroutines. The referencing environment of a function passed as a parameter is the environment from when the function was declared, which would be bound during compile time.

g. Run time. The amount of memory the program needs is the amount of stack and heap space required. The amount of that memory which is needed can vary with the input to the program, which is not known until run time.

2.

A language that uses dynamic scoping cannot do type checking at compile time. In a dynamically scoped language, binding for a name depends on the control flow of the program, which can only be determined at run time. In other words, values of different types could legitimately be assigned to the same variable in an unpredictable fashion over the course of run time, so type checking at compile time would not be possible.

A language that uses static scoping, on the other hand, could still do type checking at run time. A language with static scoping generally does type checking at compile time for the sake of minimizing run time overhead, but in principle the type information is still available and known during run time and the checking could be conducted then as well.

3.

Scheme uses static scoping. Consider this Scheme program:

( define x 1 )

( define ( f x ) ( g 1 ) )

( define ( g y ) ( + x y ) )

( f 2 )

If this program returned 3, Scheme would have to be using dynamic scoping, where the most recent active binding for x, 2, would be chosen. Instead Scheme finds the declaration of x from the outer scope, 1, which demonstrates that Scheme does indeed use static scoping.

4.

Shallow binding: 1 0 2 0 3 0 4 0

Deep binding: 1 0 ? 2 0 0 4 4

In the shallow binding case, the foo function will have set\_x and print\_x access foo’s local x, which will be set to the specified integer n and then printed. When outside foo the global x will be used for the set\_x and print\_x functions. This produces the output above.

In the deep binding case, the foo function is designed to have set\_x access the local x when the provided n equals 1 or 3, and the global x otherwise. Foo will then use print\_x to print the local x when n equals 1 or 2, and the global x otherwise. The print\_x and set\_x calls outside of the foo function use the global x. This ends up producing the output shown above.