Bindings and Scopes Assignment

# 1. Some programming languages are typeless. What are the obvious advantages and disadvantages of having no types in a language?

The obvious advantage of having no types in a language is that any variable can be used for any data type. This makes the language flexible, and programs can be more generic. This is an advantage because a program can be written for use with several different data types. A typless language is also easier to learn and changes can be made faster. Beginners will find the lack of type declarations easier to manage. The simplicity of the lack of types can make learning how to program easier. Changes can also be made faster because of the lack of type. When modifying sections of code, programmers do not have to consider the data types, which can allow fast changes. These advantages also help make the language more writable. The disadvantage of typeless languages is that there is a decrease in readability. If any variable can be used for any type, this can lead to confusion when reading the code. This is especially true if poor variable names are chosen. Typeless languages may also be more error prone because assignments are not restricted based on type. This may lead to errors that are not caught during compilation due to the lack of typing, which only become apparent as run-time errors. For instance, if a programmer incorrectly assigns the wrong data types to the wrong variable, a typless language would not detect the error until it becomes apparent during execution.

# 2. Describe a situation when a history-sensitive variable in a subprogram is useful.

History sensitive variables are variables that retain their values between sperate executions of a certain subprogram. This is accomplished through the use of locally scoped static variables. These are local variables that are bound to the same memory cells from the beginning of program execution until the end of program execution. This means that they will not be deallocated once their scope ends. This allows for locally scoped static variables to retain their values between separate subprogram executions. A simple situation where history-sensitive variables in a subprogram would be useful is if one wanted to have a variable in a subprogram that contained an integer representing the number of times the subprogram was executed. Additionally, history-sensitive variables would be useful in a pseudo random number generator subprogram because the generator could use the last number it generated to generate the next pseudo random number. The history sensitive variable would keep the last number generated and upon the next call to the random number generator, the history sensitive variable could be used to generate the next number. This is one type of implementation that pseudo random number generators use. Coroutines and subprograms used in iterator loops are two more situations in which history sensitive variable could be useful.

# 3. Consider the following Ada skeletal program. List all the variables, along with the program unit where they are declared, t­­­­hat are visible in the bodies of Sub1, Sub2, and Sub3. Assuming static scoping is used.

Text, letter

Description automatically generated

|  |  |  |
| --- | --- | --- |
| In Sub1 | In Sub2 | In Sub3 |
| X - Main  A, Y, Z - Sub1 | X - Main  Y - Sub1  A, B, Z - Sub2 | Y, Z - Main  A, X, W – Sub3 |

# 4. Consider the following skeletal C program. Given the following calling sequences and assuming that dynamic scoping is used, what variables are visible during execution of the last function called? Include with each visible variable the name of the function in which it was defined.

A picture containing graphical user interface

Description automatically generated

## a. main calls fun1; fun1 calls fun2; fun2 calls fun3

## b. main calls fun1; fun1 calls fun3

## c. main calls fun2; fun2 calls fun3; fun3 calls fun1

## d. main calls fun3; fun3 calls fun1

|  |  |  |
| --- | --- | --- |
|  | **Variable** | **Where declared** |
| a. | a  b  c  d, e, f | main  fun1  fun2  fun3 |
| b. | a  b, c  d, e, f | main  fun1  fun3 |
| c. | a  b, c, d  e, f | main  fun1  fun3 |
| d. | a  b, c, d  e, f | main  fun1  fun3 |