In Fortran, initial values of variables can be specified in a data statement as in

```
Real PI
Integer SUM
Data SUM /0/, PI /3.14159/
```

which initializes `SUM` to 0 and `PI` to `3.14159`. The actual initializations take place at compile time, in this case. Once execution begins, `SUM` and `PI` are independent of any other variables.

In many languages, initial values of variables can be specified in the declaration statement, as in the Java declaration

```
int sum = 0;
```

In general, initialization occurs only once for static variables, but it occurs with every allocation for dynamically allocated variables, such as the local variables in a Java method.

**Summary**

Case sensitivity and the relationship of names to special words, which are either reserved words or keywords, are the design issues for names.

Variables can be characterized by the sextuple of attributes: name, address, value, type, lifetime, scope.

Aliases are two or more names bound to the same storage address. They are regarded as detrimental to reliability, but are difficult to eliminate entirely from a language.

Binding is the association of attributes with program entities. Knowledge of the binding times of attributes to entities is essential to understanding the semantics of programming languages. Binding can be static or dynamic. Declarations, either explicit or implicit, provide a means of specifying the static binding of variables to types. In general, dynamic binding allows greater flexibility but at the expense of readability, efficiency, and reliability.

Scalar variables can be separated into four categories by considering their lifetimes. These are static, stack dynamic, explicit heap dynamic, and implicit heap dynamic.

Strong typing is the concept of requiring that all type errors be detected. The advantage of strong typing is increased reliability.

The type compatibility rules of a language have an important effect on the operations provided for the values in the language. Type compatibility is generally defined in terms of name compatibility or structure compatibility.

Static scoping is a central feature of ALGOL 60 and some of its descendants. It provides an efficient method of allowing visibility of nonlocal variables in subprograms. Dynamic scoping provides more flexibility than static scoping but, again, at the expense of readability, reliability, and efficiency.

The referencing environment of a statement is the collection of all of the variables that are visible to that statement.

Named constants are simply variables that are bound to values only when they are bound to storage. Initialization is the binding of a variable to a value at the time the variable is bound to storage.

**Review Questions**

1. What are the design issues for names?
2. What is the potential danger of case-sensitive names?
3. In what way are reserved words better than keywords?
4. What is an alias?
5. Which category of C++ reference variables are always aliases?
6. What is the `r-value` of a variable? What is the `r-value`?
7. Define binding and binding time.
8. After language design and implementation, what are the four times bindings can take place in a program?
9. Define static binding and dynamic binding.
10. What are the advantages and disadvantages of implicit declarations?
11. What are the advantages and disadvantages of dynamic type binding?
12. Define static, stack-dynamic, explicit heap-dynamic, and implicit heap-dynamic variables. What are the advantages and disadvantages of these?
13. Define coercion, type error, type checking, and strong typing.
14. Define name type compatibility and structure type compatibility. What are the relative merits of these two?
15. What is the difference between an Ada derived type and an Ada subtype?
17. How is a reference to a nonlocal variable in a static-scoped program connected to its definition?
18. What is the general problem with static scoping?
19. What is the referencing environment of a statement?
20. What is a static ancestor of a subprogram? What is a dynamic ancestor of a subprogram?
21. What is a block?
22. What are the advantages and disadvantages of dynamic scoping?
23. What are the advantages of named constants?
1. Decide which of the following identifier forms is most readable, and then support that decision.

\begin{verbatim}
SubOfSales
sum_of_sales
SUNOfSALES
\end{verbatim}

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

3. One common use of Fortran's `equivalence` is the following: A large array of numeric values is made available to a subprogram as a parameter. The array contains many different unrelated variables, rather than a collection of repetitions of the same variable. It is represented as an array to reduce the number of names that need to be passed as parameters. Within the subprogram, a `lengthy equivalence` statement is used to create connotative names as aliases to the various array elements, which increases the readability of the code of the subprogram. Is this a good idea or not? What alternatives to aliasing are available?

4. Write a simple assignment statement with one arithmetic operator in some language you know. For each component of the statement, list the various bindings that are required to determine the semantics when the statement is executed. For each binding, indicate the binding time used for the language.

5. Dynamic type binding is closely related to implicit heap-dynamic variables. Explain this relationship.

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

7. Look up the definition of `strongly typed` as given in Gelani (1983) and compare it with the definition given in this chapter. How do they differ?

8. Consider the following Ada skeletal program:

\begin{verbatim}
procedure Main is
  X : Integer;
  procedure Sub3; -- This is a declaration of Sub3
    -- It allows Sub1 to call it
  procedure Sub1 is
    X : Integer;
    procedure Sub2 is
      begin -- of Sub2
      ... end; -- of Sub2
    begin -- of Sub1
    ... end; -- of Sub1
end; -- of Sub1
procedure Sub3 is
  begin -- of Sub3
  ...
end; -- of Sub3
begin -- of Main
  ...
end; -- of Main
\end{verbatim}

Assume that the execution of this program is in the following unit order:

Main calls Sub1
Sub1 calls Sub2
Sub2 calls Sub3

a. Assuming static scoping, which declaration of \texttt{x} is the correct one for a reference to \texttt{x} in the following:

i. Sub1
ii. Sub2
iii. Sub3

b. Repeat part \( a \), but assume dynamic scoping.

9. Assume the following Ada program was compiled and executed using static scoping rules. What value of \texttt{x} is printed in procedure \texttt{Sub1}?

Under dynamic scoping rules, what value of \texttt{x} is printed in procedure \texttt{Sub1}?

\begin{verbatim}
procedure Main is
  X : Integer;
procedure Sub1 is
  begin -- of Sub1
  Put(X);
end; -- of Sub1
procedure Sub2 is
  X : Integer;
  begin -- of Sub2
  X := 10;
  Sub1
  end; -- of Sub2
begin -- of Main
  X := 5;
  Sub2
end; -- of Main
\end{verbatim}
10. Consider the following program:

```
procedure Main is
  X, Y, Z : Integer;
procedure Sub1 is
  A, Y, Z : Integer;
procedure Sub2 is
  A, B, Z : Integer;
begin  -- of Sub2
  ...
end;  -- of Sub2
begin  -- of Sub1
  ...
end;  -- of Sub1
procedure Sub3 is
  A, X, W : Integer;
begin  -- of Sub3
  ...
end;  -- of Sub3
begin  -- of Main
  ...
end;  -- of Main
```

List all the variables, along with the program units where they are declared, that are visible in the bodies of Sub1, Sub2, and Sub3, assuming static scoping is used.

11. Consider the following program:

```
procedure Main is
  X, Y, Z : Integer;
procedure Sub1 is
  A, Y, Z : Integer;
begin  -- of Sub1
  ...
end;  -- of Sub1
procedure Sub2 is
  A, X, W : Integer;
procedure Sub3 is
  A, B, Z : Integer;
begin  -- of Sub3
  ...
end;  -- of Sub3
begin  -- of Sub2
  ...
end;  -- of Sub2
begin  -- of Sub1
  ...
end;  -- of Sub1
procedure Sub3 is
  A, X, W : Integer;
begin  -- of Sub3
  ...
end;  -- of Sub3
begin  -- of Main
  ...
end;  -- of Main
```

List all the variables, along with the program units where they are declared, that are visible in the bodies of Sub1, Sub2, and Sub3, assuming static scoping is used.

12. Consider the following C program:

```
void fun(void) {
  int a, b, c; /* definition 1 */
  ...
  while (...) {
    int b, c, d; /* definition 2 */
    ...
    while (...) {
      int c, d, e; /* definition 3 */
      ...
      ...
      ...
      ...
    }
    ...
    ...
  }
}
```

For each of the four marked points in this function, list each visible variable, along with the number of the definition statement that defines it.

13. Consider the following skeletal C program:

```
void fun1(void); /* prototype */
void fun2(void); /* prototype */
void fun3(void); /* prototype */
void main() {
  int a, b, c;
  ...
  void fun1(void) {
    int b, c, d;
    ...
  }
  void fun2(void) {
    int c, d, e;
    ...
  }
  void fun3(void) {
    int d, e, f;
    ...
  }
  ...
}
```

For each of the four marked points in this function, list each visible variable, along with the number of the definition statement that defines it.
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. 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.
e. main calls fun1; fun1 calls fun3; fun3 calls fun2.
f. main calls fun3; fun3 calls fun2; fun2 calls fun1.

14. Consider the following program:

```plaintext
procedure Main is
  X, Y, Z : Integer;
procedure Sub1 is
  A, Y, Z : Integer;
  begin -- of Sub1
    ...
  end; -- of Sub1
procedure Sub2 is
  A, B, Z : Integer;
  begin -- of Sub2
    ...
  end; -- of Sub2
procedure Sub3 is
  A, X, W : Integer;
  begin -- of Sub3
    ...
  end; -- of Sub3
begin -- of Main
  ...
end; -- of Main
```

Given the following calling sequences and assuming that dynamic scoping is used, what variables are visible during execution of the last subprogram activated? Include with each visible variable the name of the unit where it is declared.

a. Main calls Sub1; Sub1 calls Sub2; Sub2 calls Sub3.
b. Main calls Sub1; Sub1 calls Sub3.
c. Main calls Sub2; Sub2 calls Sub3; Sub3 calls Sub1.
d. Main calls Sub3; Sub3 calls Sub1.
e. Main calls Sub1; Sub1 calls Sub3; Sub3 calls Sub2.
f. Main calls Sub3; Sub3 calls Sub2; Sub2 calls Sub1.

---

1. Perl allows both static and a kind of dynamic scoping. Write a Perl program that uses both and clearly shows the difference in effect of the two. Explain clearly the difference between the dynamic scoping described in this chapter and that implemented in Perl.

2. Write a COMMON LISP program that clearly shows the difference between static and dynamic scoping.

3. Write a JavaScript script that has subprograms nested three deep and in which each nested subprogram references variables defined in all of its enclosing subprograms.

4. Write a C function that includes the following sequence of statements:

```c
x = 21;
int x;
```

Run the program and explain the results. Rewrite the same code in C++ and Java and compare the results.