2.3 BINDING

A source file has many names whose properties need to be determined. The meaning of these properties might be determined at different phases of the life cycle of a program. Examples of such properties include the set of values associated with a type; the type of a variable; the memory location of the compiled function; the value stored in a variable, and so forth. **Binding is the act of associating properties with names. Binding time is the moment in the program's life cycle when this association occurs.**

Many properties of a programming language are defined during its creation. For instance, the meaning of key words such as while or for in C, or the size of the integer data type in Java, are properties defined at language design time. Another important binding phase is the language implementation time. The size of integers in C, contrary to Java, were not defined when C was designed. This information is determined by the implementation of the compiler. Therefore, we say that the size of integers in C is determined at the language **implementation time.**

Many properties of a program are determined at **compilation time**. Among these properties, the most important are the types of the variables in statically typed languages. Whenever we annotate a variable as an integer in C or Java, or whenever the compiler infers that a variable in Haskell or SML has the integer data type, this information is henceforward used to generate the code related to that variable. The location of statically allocated variables, the layout of the [activation records] of function and the control flow graph of statically compiled programs are other properties defined at compilation time.

If a program uses external libraries, then the address of the external functions will be known only at link time. It is in this moment that the runtime environment finds where is located the printf function that a C program calls, for instance. However, the absolute addresses used in the program will only be known at loading time. At that moment we will have an image of the executable program in memory, and all the dependences will have been already solved by the loader.

Finally, there are properties which we will only know once the program executes. The actual values stored in the variables is perhaps the most important of these properties. In dynamically typed languages we will only know the types of variables during the execution of the

program. Languages that provide some form of late binding will only lets us know the target of a function call at runtime, for instance

In general **binding** is the association of attribute to its entity or operation to its symbol

Time of binding is called as binding time (important in the semantics of PL's)

Binding times

1. Language design time

- is bound to the multiplication operation,
- pi=3.14159 in most PL's.

2. Language implementation time

- A data type, such as int in C is bound to a range of possible values

3. Compile time

- A Java variable is bound to its type.

4. Link time

- A call to the library subprogram is bound to the subprogram code.

5. Load time

- A variable is bound to a specific memory location.

6. Run time

- A variable is bound to a value through an assignment statement.
- A local variable of a Pascal procedure is bound to a memory location.

Example:

count = count + 5

• The type of count is bound at compile time

• The set of possible values of count is bound at compiler design time

• The meaning of the operator symbol + is bound at compile time, when the types of its operands have been determined

• The internal representation of the literal 5 is bound at compiler design time

• The value of count is bound at execution times with this Statement

Binding of attributes to variables

1. Static: if binding occurs before runtime and remains unchanged throughout the program execution.

2. Dynamic: if binding occurs during runtime or can change in the course of program execution

Type bindings

Before a variable can be referenced in a program it must be bound to a data type. It is important to identify how the type is specified and when the binding takes place

Variable declarations

1. Explicit declaration (by statement)

It is a statement in a program that lists variable names and specifies that they are a particular type

2. Implicit declaration (by first appearance)

It means of associating variables with types through default conventions, rather than declaration statements. First appearance of a variable name in a program constitutes its implicit declaration

- Both Declarations creates static binding to types.Most current PLs require explicit declarations of all variables, Exceptions are Perl, Javascript, ML Languages.
- Early languages (Fortran, BASIC) have implicit declarations
- e.g. In Fortran, if not explicitly declared, an identifier starting with I,J,K,L,M,N are implicitly declared to integer, otherwise to real type
- Implicit declarations are not good for reliability and writability because misspelled identifier names cannot be detected by the compiler

- e.g. In Fortran variables that are accidentally left undeclared are given default types, and leads to errors that are difficult to diagnose
- Some problems of implicit declarations can be avoided by requiring names for specific types to begin with a particular special characters
    - e.g. In Perl

        $apple : scalar

        @apple: array

        %apple: hash

Dynamic type binding

Type of a variable is not specified by a declaration statement, nor can it be determined by the spelling of its name

• Type is bound when it is assigned a value by an assignment statement.

• Advantage: Allows programming flexibility.

example languages : Javascript and PHP

• e.g. In JavaScript

list = [10.2 5.1 0.0]

list is a single dimensioned array of length 3.

List = 73

list is a simple integer

Disadvantage:

1. less realiable : compiler cannot check and enforce types.

Example:

Suppose I and X are integer variables, and Y is a floating-point.

The correct statement is

I := X

But by a typing error

I := Y

is typed. In a dynamic type binding language, this error cannot be detected by the compiler. I is changed to float during execution. The value of I becomes erroneous.

Disadvantage:

1. Cost:

- Type checking must be done at run-time.
- Every variable must have a descriptor to maintain current type.
- The correct code for evaluating an expression must be determined during execution.
- Languages that use dynamic type bindings are usually implemented as interpreters (LISP is such a language).

Type Inference

ML is a PL that supports both functional and imperative programming.

In ML, the type of an expression and a variable can be determined by the type of a constant in the expression without requiring the programmer to specify the types of the variables

Examples

fun circum (r) = 3.14 *r*r; (circum is real)

fun times10 (x) = 10*x; (times10 is integer)

• Note: fun is for function declaration

fun square (x) = x*x;

– Default is int. if called with square(2.75) it would cause an error

It could be rewritten as:

fun square (x: real) = x*x;

```
fun square (x):real = x*x;

fun square (x) = (x:real)*x;

fun square (x) = x*(x:real);
```

Storage Bindings and Lifetime

Allocation: process of taking the memory cell to which a variable is bound from a pool of available memory

Deallocation: process of placing the memory cell that has been unbound from a variable back into the pool of available memory

Lifetime of a variable: Time during the variable is bound to a specific memory location

According to their lifetimes, variables can be separated into four categories:

- static,
- stack-dynamic,
- explicit heap-dynamic,
- implicit dynamic.

Static Variables

- Static variables are bound to memory cells before execution begins, and remains bound to the same memory cells until execution terminates.
- Applications: globally accessible variables, to make some variables of subprograms to retain values between separate execution of the subprogram
- Such variables are history sensitive.

Advantage: Efficiency. Direct addressing (no run-time overhead for allocation and deallocation).

Disadvantage: Reduced flexibility.

• If a PL has only static variables, it cannot support recursion.

• Examples: All variables in FORTRAN I, II, and IV

• Static variables in C, C++ and Java

Stack-Dynamic Variables

- Storage binding: when declaration statement is elaborated (in run-time).
- Type binding: statical.
- Example: A Pascal procedure consists of a declaration section and a code section.The local variables get their type binding statically at compile time, but their storage binding takes place when that procedure is called. Storage is deallocated when the procedure returns.,
- Local variables in C functions.
- Advantages: Dynamic storage allocation is needed for recursion. Same memory cells can be used for different variables (efficiency)
- Disadvantages: Runtime overhead for allocation and deallocation
- In C and C++, local variables are, by default, stack-dynamic, but can be made static through static qualifier.

```
foo ()

{

static int x;
        …}
```

Explicit Heap-Dynamic Variables

- Nameless variables
- Storage allocated/deallocated by explicit run-time instructions
- can be referenced only through pointer variables
- types can be determined at run-time
- storage is allocated when created explicitly
- **Advantages:** Required for dynamic structures (e.g., linked lists, trees)
- **Disadvantages**: Difficult to use correctly, costly to refer, allocate, deallocate.

Implicit Heap-Dynamic Variables

• Storage and type bindings are done when they are assigned values.

• **Advantages**: Highest degree of flexibility

• **Disadvantage**s:

- Runtime overhead for allocation and deallocation
- Loss of error detection by compiler
- Examples: Javascript and APL variables.