Possible Binding Times

• Language design time -- bind operator symbols to operations

• Language implementation time-- bind floating point type to a representation

• Compile time -- bind a variable to a type in C or Java

• Load time -- bind a C or C++ static variable to a memory cell)

• Runtime -- bind a nonstatic local variable to a memory cell

Example: Java Assignment Statement

• In the simple statement count = count + 1;

• Some of the bindings and binding times are

- Type of count is bound at compile time
- Set of possible values is bound at design time (but in C, at implementation time)
- Meaning of + is bound at compile time, when types of operands are known
- Internal representation of the literal 1 is bound at compiler design time
- Value of count is bound at runtime with this statement

Static Storage Binding:

Static variables are bound to memory cells before program execution begins and remain bound to those same memory cells until program execution terminates.

e.g., **globally accessible variables** (FORTRAN I, II and IV)
e.g., C, C++, and Java include the **static** specifier on local variable definition
e.g., Pascal does not provide static variables.

Advantage: history sensitive
Disadvantage: no recursion supported

```
//extract a character from the string.
//static will remember last position.
char lexical()
{
     static int position = -1;
     position = position + 1;
     return text[position];
}
```

Static variables are bound to addresses before execution begins and remain bound until program terminates:

- Most global variables are static
- Some languages such C/C++ support local static variables

Advantages:

- efficiency (direct addressing), history-sensitive subprogram support

Disadvantage: lack of flexibility

- A language with ONLY static variables does not support recursion

Examples:

In C and C++ a variable declared as static inside a function retains its value over function invocations:

```
int hitcount() {
     static int count  = 0;
     return ++count;
}
```

But inside C#,C++ and Java class definitions, the static modifier means that the variable is a class variable rather than an instance variable.

Global Variables in Java

• Java does not allow declaration of variables outside of any scope, so C/C++ style globals cannot be used

• Solution is to use public static variables in a class

```
public class GlobalData {
     public static int usercount = 0;
     public static long hitcount = 0;
}
```

Stack-Dynamic Storage Binding:

Stack-dynamic variables are bound to storages when execution reaches the code to which the declaration is attached. (But, data types are statically bound.) That is, stack-dynamic variables are allocated from the **run-time stack**.

e.g., A Pascal procedure consists of a declaration section and a code section.
e.g., FORTRAN 77 and FORTRAN 90 use SAVE list for stack-dynamic list.
e.g., C and C++ assume local variables are static-dynamic.

Recursive subprograms: each call of subprogram has its own local storage.

Storage bindings are created for variables when their declaration statements are elaborated.

- A declaration is elaborated when the executable code associated with it is executed

For scalar variables, all attributes except address are statically bound

- local variables in C subprograms and Java methods
- Note that while actual memory address is not statically bound a relative address (stack offset) is statically bound

Advantage: allows recursion; conserves storage

Disadvantage:

- Overhead of allocation and deallocation
- Subprograms cannot be history sensitive
- Inefficient references (indirect addressing)

```
#include <iostream.h>

void Recursion(int n)
{
     if(n > 0) {
          n--;
          Recursion(n);
          cout<<n<<" ";
     }
}

void main() {
     Recursion(3);
}
```

Heap Dynamic:

In programming, heap refers to a common pool of memory that is available to the program. The management of the heap is either done by the applications themselves, allocating and deallocating memory as required, or by the operating system or other system program.

Explicit Heap-Dynamic Storage Binding:

Explicit Heap-Dynamic variables are nameless (abstract) memory cells that are allocated and deallocated by explicit run-time instructions specified by the programmer.

- Allocated and deallocated by explicit directives, specified by the programmer, which take effect during execution
- Referenced only through pointers or references, e.g. dynamic objects in C++ (via new and delete), **all objects in Java**, malloc and free in C

```
int *intnode; //c or c++
intnode = new int;
. . .
delete intnode;
```

Example in C++:

```
int* intnode;
...
intnode = new int; // allocates an int cell
...
delete intnode; // deallocates the cell to which intnode
                   points
```

The new operator automatically creates an object of proper size, and remains a pointer of the connect type. To free the space for this object, use the delete operator. (With explicit heap dynamic variables we do not need to incur the overhead of garbage collection).

Disadvantages:

1. Failure to deallocate storage results in memory leaks
2. Other difficulties in using variables correctly (dangling pointers, aliasing)
3. Heap fragmentation

On page 229 of Data Structures with C++ using STL, second edition, by William Ford & William Topp:

> ***You can think of the heap as a bank of memory, much like a financial bank that maintains a reserve of money. A program can borrow (allocate) memory from the heap when additional storage space is required and then pay it back (deallocate) when it is no longer needed.***

Implicit Heap-Dynamic Storage Binding:

Implicit Heap-Dynamic variables are bound to heap storage only when they are assigned values. It is similar to dynamic type binding.

Implicit heap-dynamic—bound to heap storage by by assignment statements:

- All attributes are bound by at this time.
- all variables in APL; all strings and arrays in Perl, JavaScript, and PHP.

Names are in effect just holders for pointers.

Advantage: flexibility (generic code)

Disadvantages:

- Inefficient, because all attributes are dynamic
- Loss of error detection

Example:

The example once again from Robert Sebesta's Concepts of Programming Languages 9th edition:

```
highs = [74, 84, 86, 90, 71];
```

This Javascript array is only brought into being when its assigned values. It then gets placed in the heap as a collection of those values.

It would bind those values to that variable. It would not necessarily tell you what type they were. That part will be dependent on other language semantics.