**Section 3.1**

**Static versus dynamic.**

Static data bind before run time dynamic data bind at runtime

**What is meant by binding?**

The association of a name with is content

**Know compile time, link time, load time, run time.**

**- compile time**

Compilers choose the mapping of high-level constructs to machine

code, including the layout of statically defined data in memory.

- **link time**

Since most compilers support *separate compilation*—compiling different

modules of a program at different times—and depend on the availability

of a library of standard subroutines, a program is usually not complete

until the various modules are joined together by a linker. The linker chooses

the overall layout of the modules with respect to one another. It also resolves

intermodule references.When a name in one module refers to an object in another

module, the binding between the two was not finalized until link time.

**- load time**

Load time refers to the point at which the operating system loads the

program into memory so that it can run. In primitive operating systems, the

choice of machine addresses for objects within the program was not finalized

until load time. Most modern operating systems distinguish between virtual

and physical addresses. Virtual addresses are chosen at link time; physical addresses

can actually change at run time. The processor’s memory management

hardware translates virtual addresses into physical addresses during each individual

instruction at run time.

**Section 3.2**

**What is meant by the lifetime of a binding?**

The period of time between the creation and the destruction of a name-to object

binding is called the binding’s *lifetime*.

**What is meant by allocation?**

**What is static allocation,?**

is the allocation of memory at [compile-time](http://en.wikipedia.org/wiki/Compile_time) before the associated program is executed.

An application of this technique involves a program module (e.g. function or [subroutine](http://en.wikipedia.org/wiki/Subroutine)) declaring static data locally, such that these data are inaccessible in other modules unless references to it are passed as [parameters](http://en.wikipedia.org/wiki/Parameters_(computer_science)) or returned. A single copy of static data is retained and accessible through many calls to the function in which it is declared. Static memory allocation therefore has the advantage of modularizing data within a program design in the situation where these data must be retained through the runtime of the program.

**stack allocation?**

[Stacks](http://en.wikipedia.org/wiki/Stack_(abstract_data_type)) in computing architectures are regions of [memory](http://en.wikipedia.org/wiki/Memory_(computers)) where data is added or removed in a [last-in-first-out](http://en.wikipedia.org/wiki/LIFO_(computing)) manner.

In most modern computer systems, each [thread](http://en.wikipedia.org/wiki/Thread_(computer_science)) has a reserved region of memory referred to as its stack. When a function executes, it may add some of its state data to the top of the stack; when the function exits it is responsible for removing that data from the stack. At a minimum, a thread's stack is used to store the location of function calls in order to allow return statements to return to the correct location, but programmers may further choose to explicitly use the stack. If a region of memory lies on the thread's stack, that memory is said to have been allocated on the stack.

Because the data is added and removed in a last-in-first-out manner, stack-based memory allocation is very simple and typically faster than heap-based memory allocation (also known as [dynamic memory allocation](http://en.wikipedia.org/wiki/Dynamic_memory_allocation)). Another feature is that memory on the stack is automatically, and very efficiently, reclaimed when the function exits, which can be convenient for the programmer if the data is no longer required. If however, the data needs to be kept in some form, then it must be copied from the stack before the function exits. Therefore, stack based allocation is suitable for temporary data or data which is no longer required after the creating function exits.

**heap allocation?**

A *heap* is a region of storage in which subblocks can be allocated and deallocated

at arbitrary times.2 Heaps are required for the dynamically allocated pieces of

linked data structures and for dynamically resized objects, such as fully general

character strings, lists, and sets, whose size may change as a result of an assignment

statement or other update operation.

**What types of variables and data use these forms of allocation?**

A *heap* is a region of storage in which subblocks can be allocated and deallocated

at arbitrary times.2 Heaps are required for the dynamically allocated pieces of

linked data structures and for dynamically resized objects, such as fully general

character strings, lists, and sets, whose size may change as a result of an assignment

statement or other update operation.

What is garbage?

Dealocation of memory

What is a dangling reference?

If an object is deallocated too soon, the program

may follow a *dangling reference*, accessing memory now used by another object.