Chapter – 7
Run Time Environments

Outline

- Introduction
- Source language issues
- Storage organization
- Storage Allocation Strategies

Introduction

- Before code generation, ***static source text*** of a program needs to be related to the actions that must occur at runtime to implement the program.
- As execution proceeds, the **same name** in the source text can denote different data objects in the target machine.
- The allocation and deallocation of data objects is managed by the runtime support package.
- The design of the run-time support package is influenced by the ***semantics*** of procedures.
- Each execution of a *procedure* is referred to as an ***activation of the procedure***.

Cont…

- If the procedure is recursive, several of its activations may & alive at the same time.
- The representation of a data object at run time is determined by its ***type***.
- Often, elementary data types, such as characters, integers, and reals can be represented by equivalent data objects in the ***target machine***.
- However, aggregates, such as arrays, strings, and structures, are usually represented by collections of ***primitive*** objects.

Source Language Issues

- For specificity, suppose that a program is made up of procedures.

  ❖**Procedures**

- ✓A *procedure* definition is a declaration that associates an *identifier* with a statement (identifier is the *procedure name*, and the statement is the *procedure body).*
- ✓Procedures that return values are called *function* in many languages;
- ✓When a procedure name appears within an executable statement, we say that the procedure is *called* at that point. A procedure is *activated* when it is called.

Cont…

- ✓The basic idea is that a procedure call executes the procedure body. Note that procedure calls can also occur within expressions.
- ✓The *lifetime* of an activation of a procedure is the sequence of steps between the first and last steps in the execution of the procedure body
- ✓A procedure is *recursive* if a new activation can begin before an earlier activation of the same procedure has ended.

❖The Scope of a Declaration

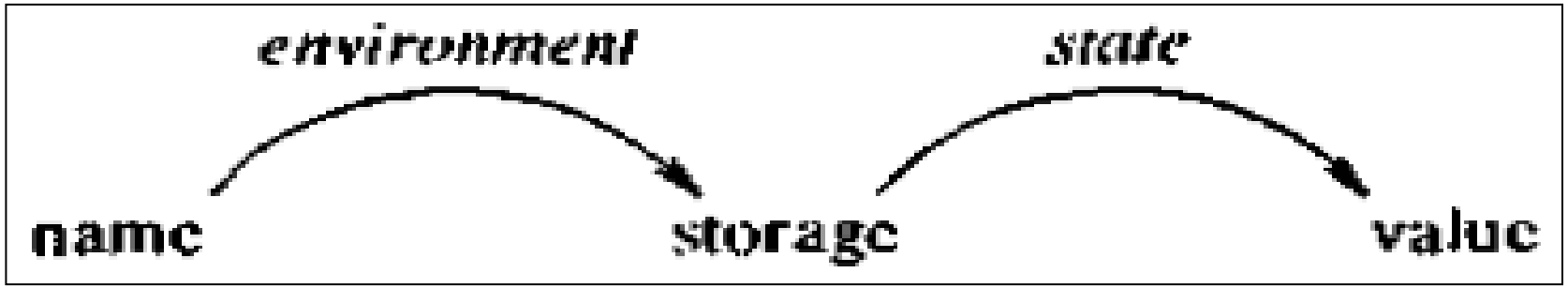
- A declaration in a language is a syntactic construct that associates information with a name.
- Declarations may be explicit or they may be Implicit.
- There may be independent declarations of the same name in different parts of a program.
- The *scope rules* of a language determine which declaration of a name applies when the name appears in the text of a program.

Cont..

- The portion of the program to which a declaration applies is called the *scope* of that declaration (*local* vs *nonlocal)*.
- At compile time, the symbol table can be used to find the declaration that applies to an occurrence of a name.
- When a declaration is seen, a symbol table entry is created for it.
- As long as we are in the scope of the declaration, its entry is returned when the name in it is looked up.

❖Binding of Names

- Even if each name is declared once in a program, the same name may denote different data objects at run time.
- The informal term "data object" corresponds to a storage location that can hold values.
- In programming language semantics, the term *environment* refers to a function that maps a name to a storage location, and the term *state* refers to a function that maps a storage location to the value held there.



Cont..

- Environments and states are different; an assignment changes the ***state***, but not the ***environment***.
- For example, suppose that storage address 100, associated with variable *pi*, holds 0. After the assignment *pi*:= 3. 14, the same storage address is associated with *pi*, but the value held there is 3.14.
- When an environment associates storage location *s* with a name *x*, we say that *x is* bound to *s*; the association itself is referred to as a *binding* of x.
- A binding is the dynamic counterpart of a declaration.

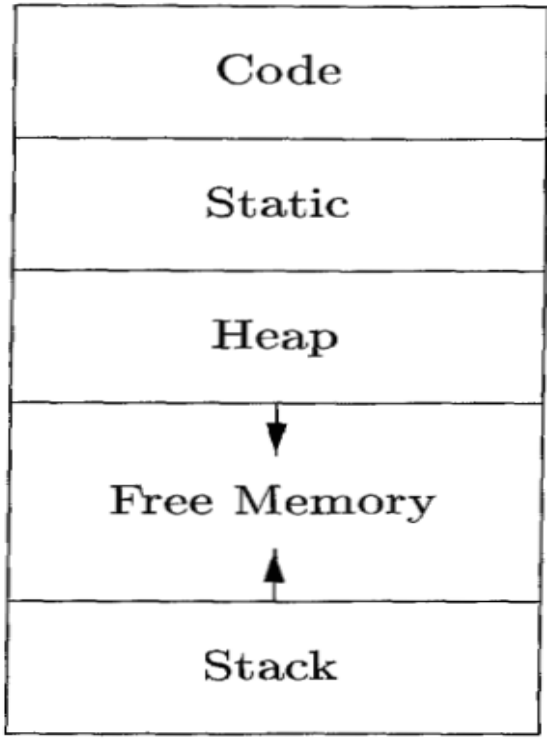
Storage Organization

❖Subdivision of Run-Time Memory

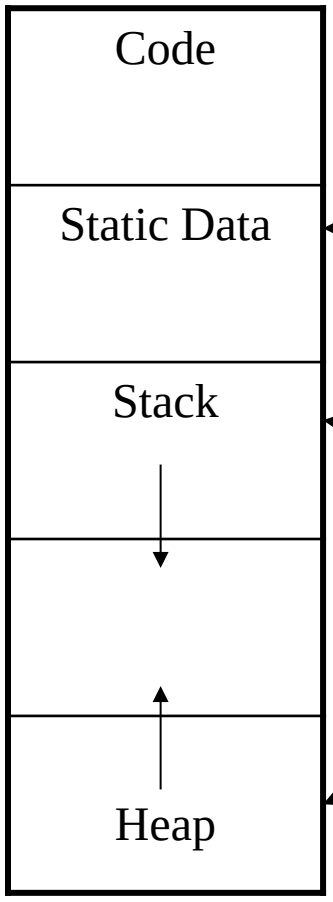
- Suppose that the compiler obtains a block of storage from the operating system for the compiled program to run in.
- This run-time storage might be subdivided to hold:
  - ✓ the generated target code,
  - ✓ data objects, and
  - ✓ control stack to keep track of procedure activations.
- The size of the generated target code is fixed at compile time, so the compiler can place the executable target code in a statically determined area *Code*.
- Similarly, the size of some program data objects, such as global constants, may be known at compile time, and these data objects can be placed in another statically determined area called ***Static***.

- To maximize the utilization of space at run time, the other two areas, *Stack* and *Heap*, are at the opposite ends of the remainder of the address space.
- These areas are dynamic; their size can change as the program executes.
- The stack is used to store data structures called activation records that get generated during procedure calls; and
- Heap holds all other runtime information.

- The sizes of the stack and the heap can change as the program executes, so we show these at opposite ends of memory, where they can grow toward each other as needed.
- The heap grows towards higher, and stacks grow down.



Run-Time Storage Organization



Memory locations for code are determined at compile time.

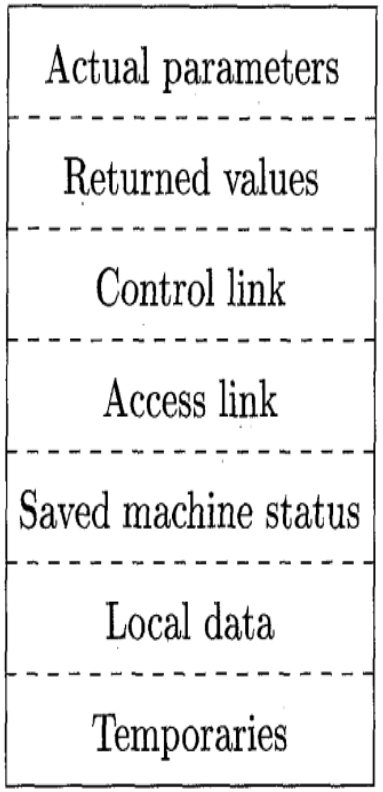
Locations of static data can also be determined at compile time.

Data objects allocated at run-time. (Activation Records)

Other dynamically allocated data objects at run-time. (For example, malloc area in C).

❖ Activation Records

- Information needed by a single execution of a procedure is managed using a contiguous block of storage called an *activation record* or *frame*.
- It consists of the collection of fields.



❑ The purpose of the fields of an activation record is as follows.

- ✓ Temporary values are stored in the field for temporaries.
- ✓ Local data holds data that is local to an execution of a procedure.
- ✓ Machine status holds information about the state of the machine just before the procedure is called.
- ✓ An access link is needed to locate data needed by the called procedure.
- ✓ The control *link* points to the activation record of the caller.
- ✓ The field for actual parameters is used by the calling procedure to supply parameters to the called procedure.
- ✓ The field for the returned value is used by the called procedure to return a value to the calling procedure.

Storage Allocation Strategies

A different storage-allocation strategy is used for organization of memory.

❖ Static Allocation

- It lays out storage for all data objects at compile time.
- Names are bound to storage as the program is compiled.
- There is no need for a run-time support package.
- Every time a procedure is activated, its names are bound to the same storage locations.
- When control returns to a procedure, the values of the locals are the same as they were when control left the last time.
- The compiler determines the amount of storage to set aside for that name from the type of a name.

❑ Some limitations go along with wing static allocation alone.

✓ The size of a data object and constraints on its position in memory must be known at compile time.

✓ Recursive procedures are restricted, because all activations of a procedure use the same bindings for local names.

✓ Data structures cannot be created dynamically, since there is no mechanism for storage allocation at run time.

❖ Stack Allocation

- Stack allocation is based on the idea of a control stack; storage is organized as a stack.
- Activation records are pushed and popped as activations begin and end, respectively.
- Storage for the locals in each call of a procedure is contained in the activation record for that call.
- Locals are bound to fresh storage in each activation, because a new activation record is pushed onto the stack when a call is made.
- The values of locals are deleted when the activation ends;
- That is, the values are lost because the storage for locals disappears when the activation record is popped.

❖ Heap Allocation

- The stack allocation strategy cannot be used for the following conditions.
  - ✓ The values of local names must be retained when an activation ends.
  - ✓ A called activation outlives the caller.
- For the above cases, the deallocation of activation records need not occur in a last-in first out fashion, so storage cannot be organized as a stack.
- Heap allocation parcels out pieces of contiguous storage, as needed for activation records or other objects.
- In heap allocation, pieces may be deallocated in any order, so over time the heap will consist of alternate areas that are free and in use.

Endof ch7…

Thank You

?