

Run-Time Environments

Introduction

In previous chapters we have studied the phases of a compiler:

- Scanning
- Parsing
- Static semantic analysis

These stages

- depend only on the properties of the source language.
- are completely independent Of
 - » the target (machine or assembly) language
 - » The properties of the target machine
 - » Operating system

Runtime Environment

1. **Runtime Environment** is the structure of the target computer's registers and memory that serves to manage memory and maintain the information needed to guide the execution process.
2. Three kinds of run time environment
 - Fully static environment (FORTRAN77)
 - Stack-based environment (C, C++, PASCAL)
 - Fully dynamic environment (LISP)
3. Hybrids of these are also possible.

Source Language Issues

1. A program is made up of procedures.
2. What are the differences between the source text of a procedure and its activation at run time?
3. A procedure definition is a declaration that associates an identifier with a statement.
 - Identifier is a procedure name
 - The statement is the procedure body
4. Procedures that return values are called function in many languages.
5. Formal and actual parameters

```
int max(int x, int y) { .....}
```

```
-----
```

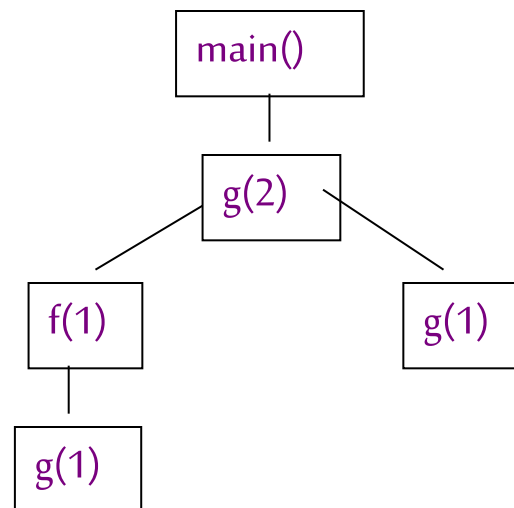
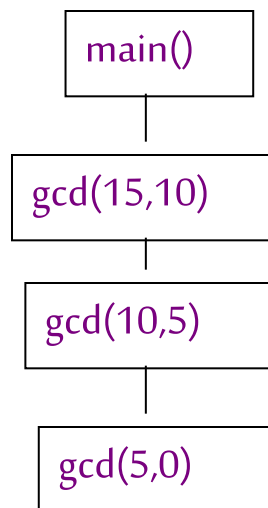
```
max(m,n)
```

Activation Trees

1. Assumptions about **flow of control** among procedures during execution of a program:
 - Control flows sequentially
 - Each execution of a procedure **starts** at the beginning of the **procedure body** and eventually **returns** to the point **immediately following** the place where the procedure was called.
2. Each execution of a procedure body is referred as **an activation** of the procedure.
3. Lifetime of an activation of a procedure P is the sequence of steps between the first and last steps in the execution of the procedure body, including time spent executing procedures called by P , the procedures called by them and so on.
4. If a and b are procedure activations, then their life times are either non-overlapping or are nested.
5. A procedure is nested if a new activation can begin before an earlier activation of the same procedure has ended.
6. A procedure is recursive if a new activation can begin before an earlier activation of the same procedure has ended.

Activation Trees (cont.)

1. We can use activation tree to depict the way control enters and leaves activations.
2. In an activation tree
 - Each node represents an activation of a procedure
 - The root represents the activation of the main program
 - The node for a is the parent of the node for b iff control flows from the activation a to b, and
 - The node for a is left of the the node for b iff the lifetime of a occurs before the lifetime of b.



Control Stacks

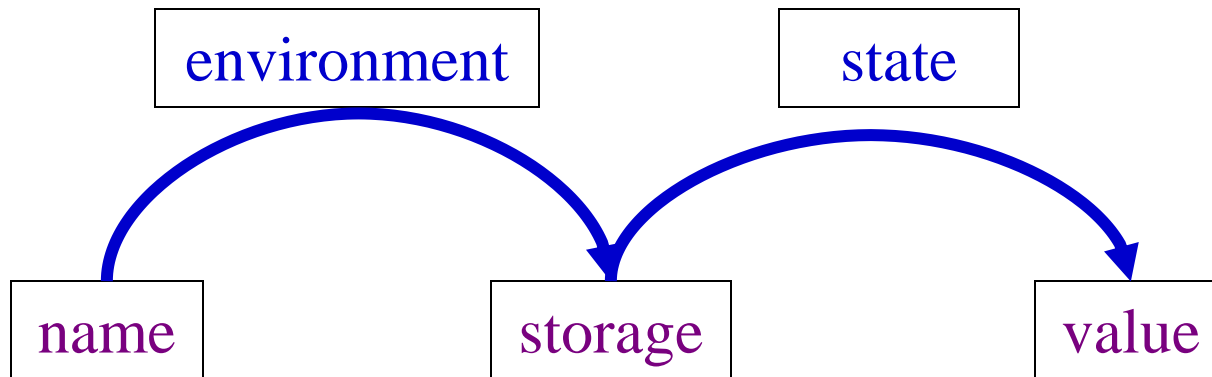
1. The flow of control in a program corresponds to a **depth-first-traversal** of the activation tree that starts at the root, visits a node before its children, and recursively visits children at each node in a **left-to-right** order.
2. We can use a stack, called **control stack** to keep track of live procedure activations.
3. The idea is to **push** the node for an activation onto the control stack as the activation begins and **pop** the node when the activation ends.
4. Thus the contents of the stack are related to **paths** to the root of the activation tree.
5. When node **n** is at the top of the control stack, the stack contains the nodes along the path from **n** to the root.

The Scope of a Declaration

1. A declaration is a syntactic construct that associates the information with a name.
2. Declaration may be
 - Explicit
 - `var i : integer`
 - Implicit
 - Any variable name starting with I is assumed to denote an integer (Fortran)
3. The scope rules determine which declaration of a name applies when the name appears in the text of a program.
4. The portion of the program to which a declaration applies is called scope of that declaration.
5. An occurrence of a name in a procedure is said to be **local** to the procedure if it is in the scope of a declaration within the procedure, otherwise, the occurrence is said to be **non-local**.
6. When a declaration is seen, a symbol table entry is created for it.

Binding of Names

1. Even if each name is declared once in a program, the same name may denote different data objects at run time.
2. Data object corresponds to a storage location that can hold values.
3. The term environment refers to a function that maps a name to a storage location.
4. The term state refers to a function that maps a storage location to the value held there.



5. Environments and states are different; an assignment changes the state but not the environment.

Binding of Names (cont.)

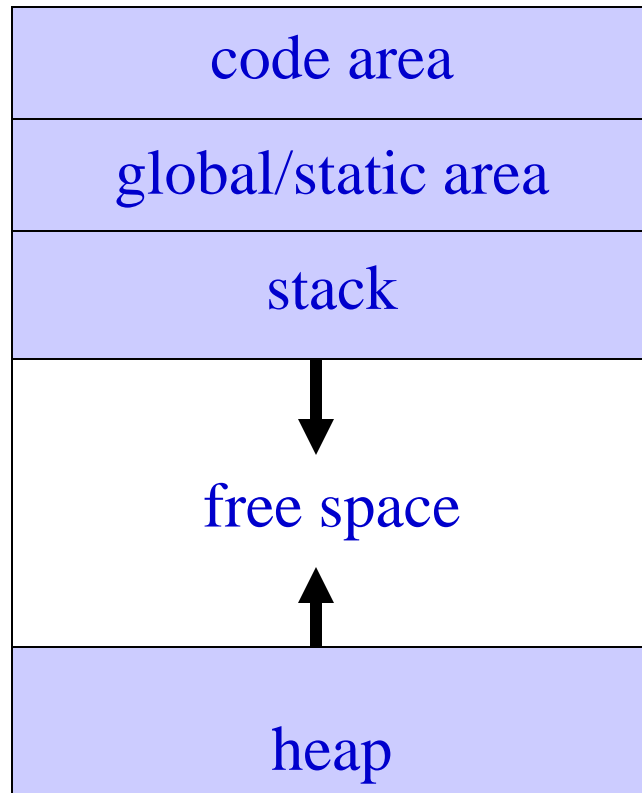
1. When an environment associates storage location **S** with a name **X**, we say that **X** is *bound* to **S**; the association itself is referred to as a *binding* of **X**.
2. A binding is the dynamic counterpart part of a declaration.

Static Notation	Dynamic Counterpart
Definition of a procedure	Activations of the procedure
Declaration of a name	Binding of the name
Scope of a declaration	Lifetime of a binding

Storage Organization

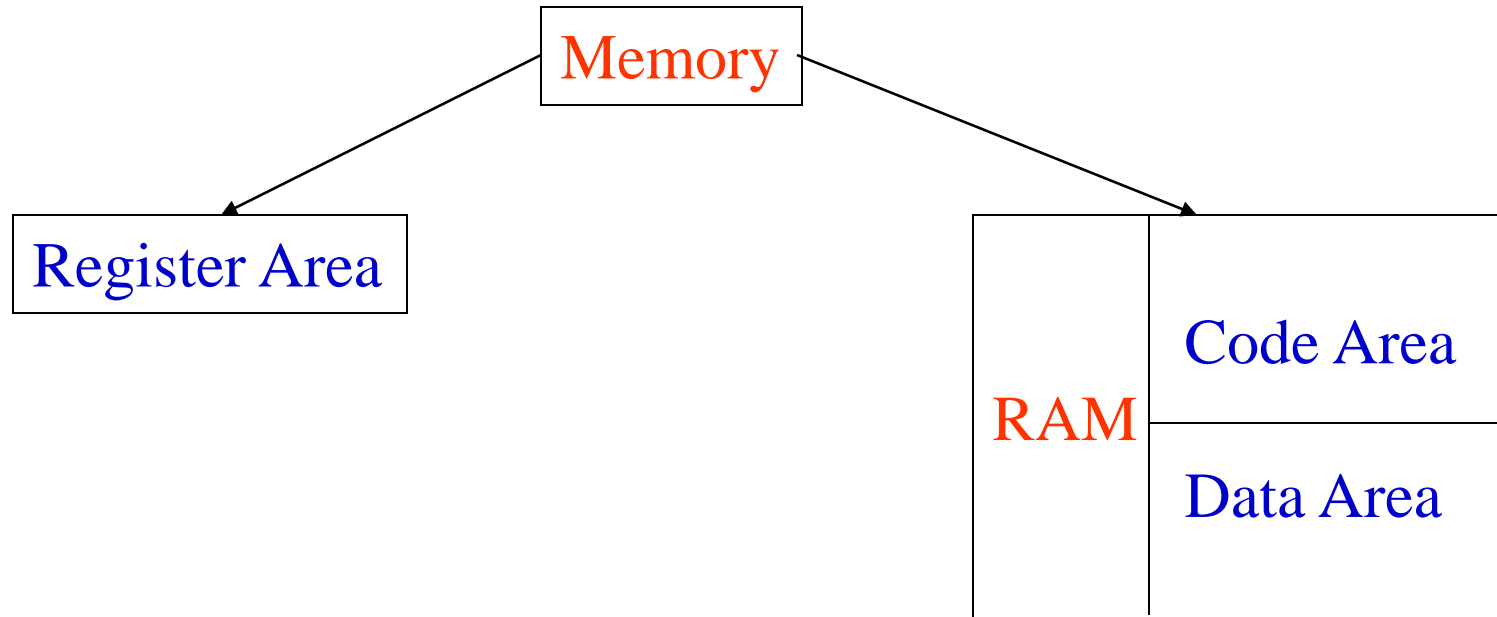
1. The compiler obtains a block of storage from the operating system for the compiled program to run in.
2. Run time storage might be subdivided to hold:
 - The generated target code
 - Data objects, and
 - A counterpart of the control stack to keep track of procedure activations.
3. In most compiled languages, it is not possible to make changes to the code area during execution.
4. The code area is fixed prior to the execution, and all code addresses are computable at compile time.
5. The size of the some data objects may also be known at compile time, and these too can be placed in a statically determined area.

Memory Organization (cont.)

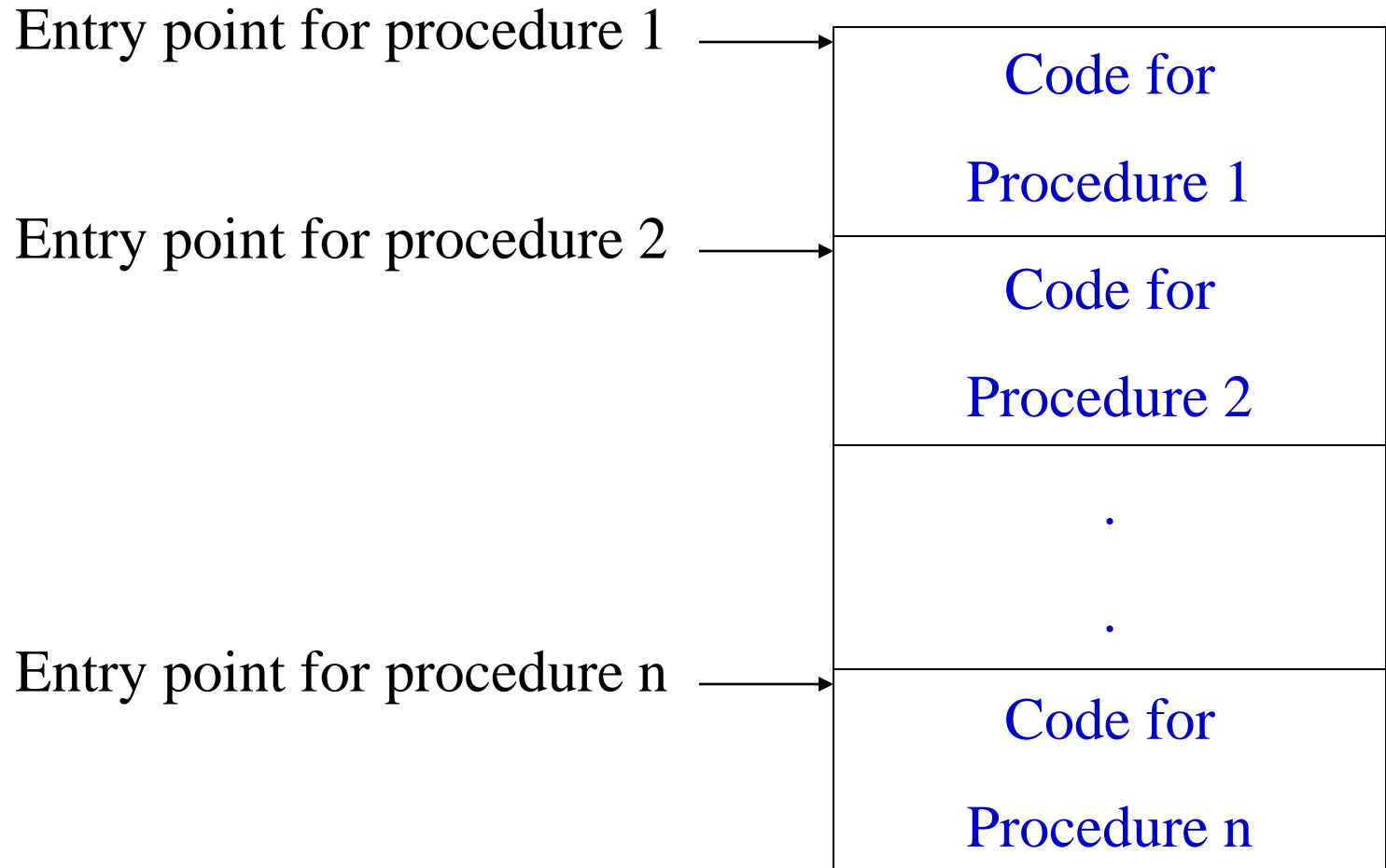


May keep information about
activation

Storage Organization Fortran



Code Memory



Entry point of each procedure and function is known at compile time.

Data Area

1. Only a small part of data can be assigned fixed locations before execution begins
 - Global and/or static data
 - Compile-time constants
 - Large integer values
 - Floating-point values
 - Literal strings

Dynamic Memory

1. The memory area for the allocation of dynamic data can be organized in many different ways.
2. A typical organization divides the dynamic memory into
 - stack area (LIFO protocol)
 - heap area

Why Stack and Heap?

1. C/Pascal uses extensions of the control stack to manage activations of procedures.
2. When a call occurs, execution of an activation is **interrupted** and information about the status of the machine, such as the value of the **program counter and machine registers**, is **saved** on the **stack**.
3. When control returns from the call, this activation can be **restarted** after **restoring** the values of relevant registers and setting the **program counter** to the point **immediately after the call**.
4. Heap holds all other information like the data under program control i.e. dynamic memory allocation.
5. By convention, stacks grow down. Heaps grow up.

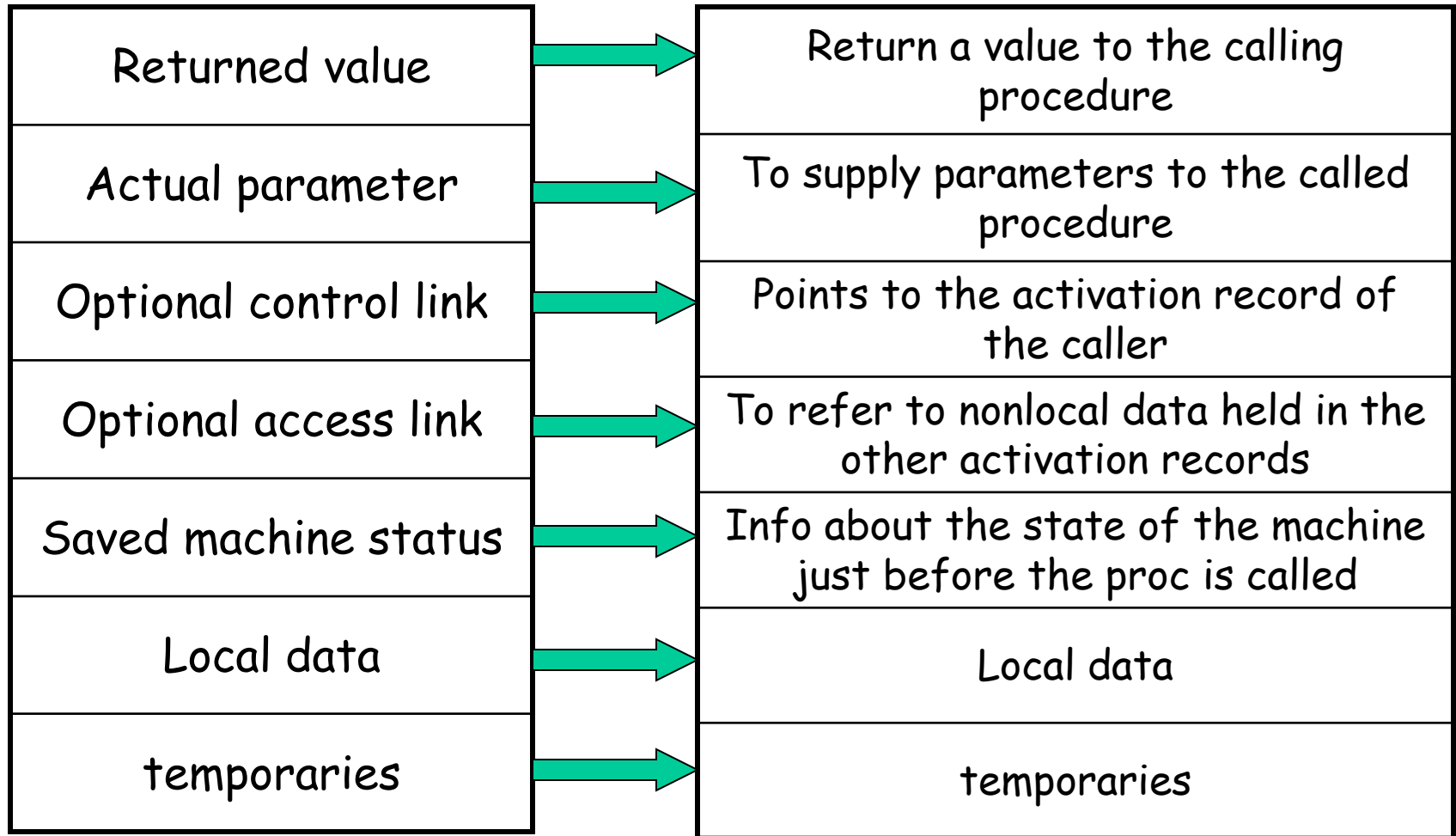
Procedure Activation Record

1. An important unit of memory
2. **Procedure activation record** contains memory allocated for the local data of a procedure or function when it is called, or activated.

arguments
bookkeeping information (return address)
local data
local temporaries

1. The picture illustrates the general organization of an activation record.
2. Details depend on the architecture of target machine and properties of the language.
3. When activation records are kept on stack, they are called **stack frames**.

Procedure Activation Record (cont.)



Storage-Allocation Strategies

1. **Static allocation** : Lays out storage for all data objects at compile time
2. **Stack allocation** : manages the run time storage as a stack
3. **Heap allocation** : allocates and deallocates storage as needed at run time from a data area known as heap.

1. Static allocation


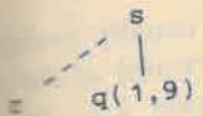
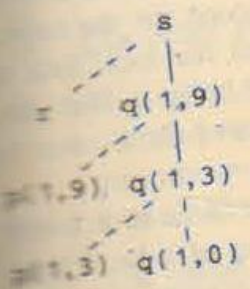
1. Names are bound to storage as the program is compiled. No need for a run time support package
2. Names are bound to the same storage allocation for every time a procedure is activated
3. Allows the values of local names to be retained across activations of a procedure i.e when control returns to a procedure, the values of the locals are the same as they were when control left the last time.
4. From the type of name, the compiler determines the amount of storage to set aside for that name
5. Hence, at the compile time we can therefore fill the addresses at which information is to be saved when procedure call occurs are also known at compile time.

Static allocation -Limitations

1. The size of data object and constraints on its position in memory must be known at compile time.
2. Recursive procedures are restricted, since all activations of a procedure use the same bindings for local names
3. Data structures can not be created dynamically, since there is no mechanism for storage allocation at run time.

2. Stack allocation

1. Storage is organized as stack
2. Activation records are Pushed and Popped for each begin and end of activation
3. Storage for locals is contained in the activation record for that call.
4. For each call locals are bound to fresh storage
5. Values for locals are deleted when activation ends.
6. Size of activation records are known at compile time.

POSITION IN ACTIVATION TREE	ACTIVATION RECORDS ON THE STACK	REMARKS
s	<div> <div>s</div> <hr/> <div>a : array</div> </div>	Frame for s <i>activation</i> <i>rescued</i>
	<div> <div>s</div> <hr/> <div>a : array</div> <hr/> <div>r</div> <hr/> <div>i : integer</div> </div>	r is activated
	<div> <div>s</div> <hr/> <div>a : array</div> <hr/> <div>q(1,9)</div> <hr/> <div>i : integer</div> </div>	Frame for r has been popped and q(1,9) pushed
	<div> <div>s</div> <hr/> <div>a : array</div> <hr/> <div>q(1,9)</div> <hr/> <div>i : integer</div> <hr/> <div>q(1,3)</div> <hr/> <div>i : integer</div> </div>	Control has just returned to q(1,3)

Calling sequences

1. Call sequence : Allocates an activation record
2. Return Sequence : Restores state of machine so that the calling procedure can continue execution.
3. Calling sequence and activation records differ even for implementation of the same language
4. Code in calling procedure is often divided into the caller and the callee
5. No exact division of run time task
6. Each call has its own actual parameters, the caller usually evaluates actual parameters and communicate them to the activation record of the callee

Calling sequences

Call sequence

1. The caller evaluates actuals
2. Caller stores return address (Status)
3. Caller increments *top_sp*
4. *The callee saves register and other status information*
5. *The callee initializes its local data and begin execution.*

Possible return sequences

1. The callee places the return value next to the activation record of the caller
2. Using the information in status field, the callee restores `top_sp` and other registers.
3. Although `top_sp` has been decremented, the caller can copy the returned values into its own activation record and use it

Variable length Data

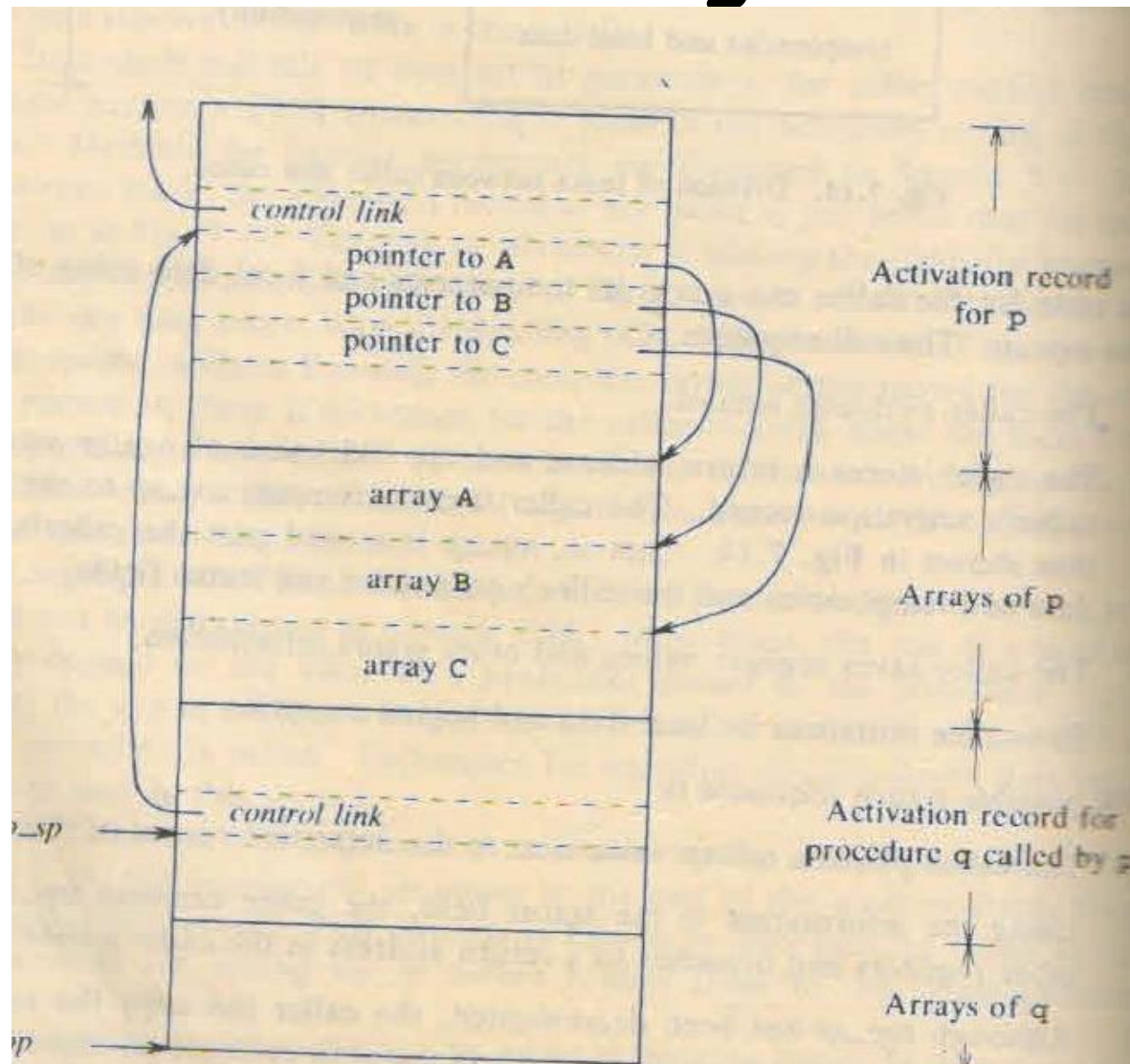


Fig. 7.15. Access to dynamically allocated arrays.

Dangling References



Heap allocation

Stack allocation can not be used when

1. The values of local names must be retained when an activation ends
2. A called activation outlives the caller. This possibility can not occur for those languages where activation tree correctly depict the flow of control between procedures.

What is Heap and Heap Management

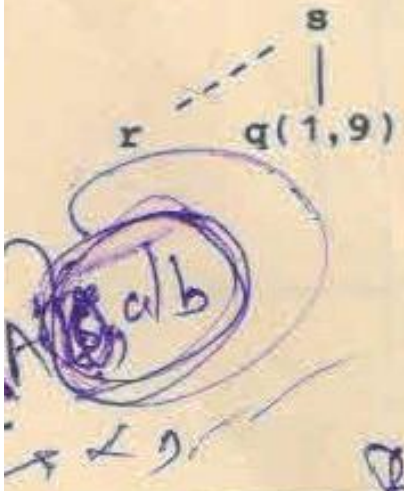
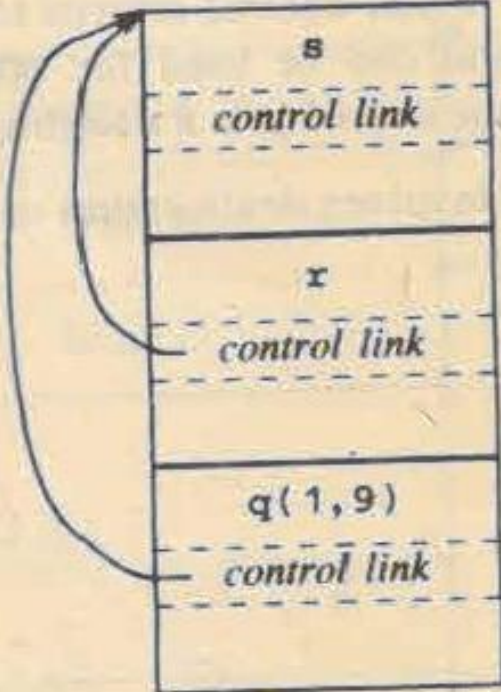
POSITION IN THE ACTIVATION TREE	ACTIVATION RECORDS IN THE HEAP	REMARKS
		Retained activation record for r

Fig. 7.17. Records for live activations need not be adjacent in a heap.

Tutorial

Explain parameter passing in detail.

Symbol Table

1. What is symbol table?
2. Symbol Table mechanisms 1) Linear list 2) Hash table
3. Performance is evaluated on the basis of
 - i. Time required to add n entries
 - ii. Time required to make e inquiries

Performance ?

Symbol Table

Symbol table entries

1. Format of entry in symbol table is not uniform

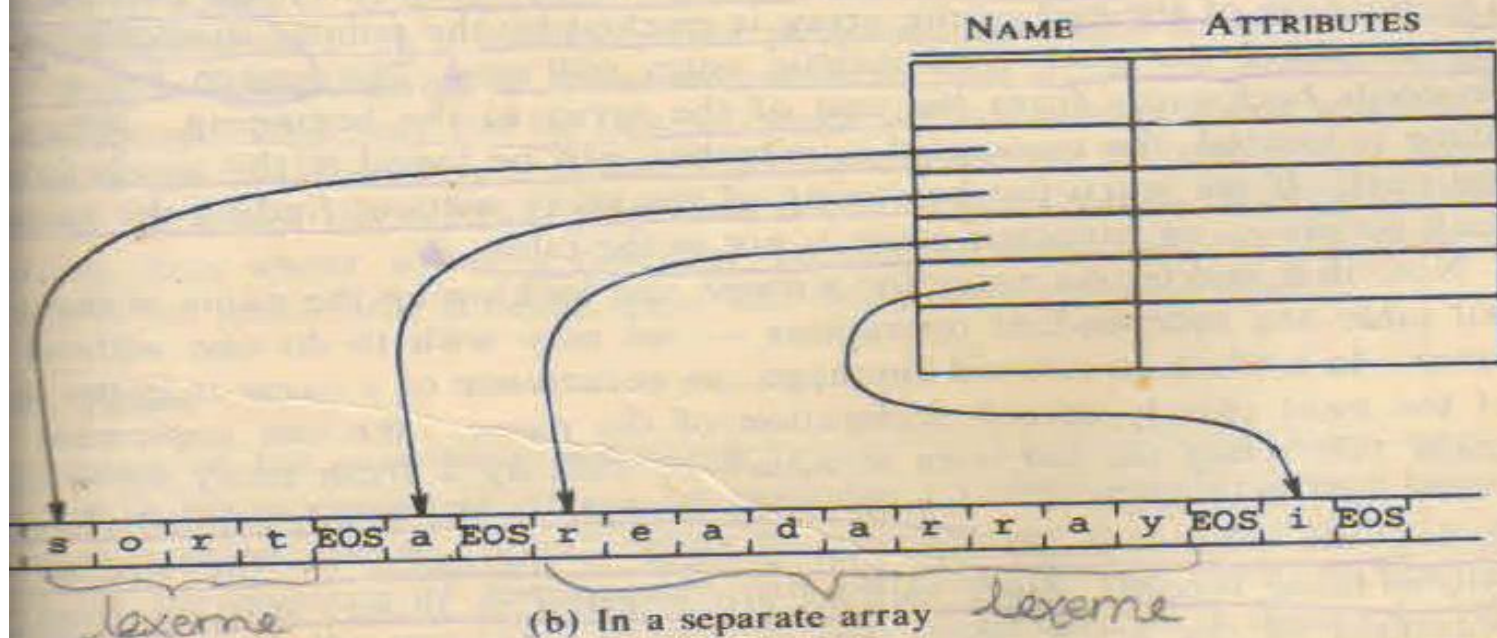
Storage allocation Information

1. Information about storage locations that will be bound to name at run time is kept in the symbol table.

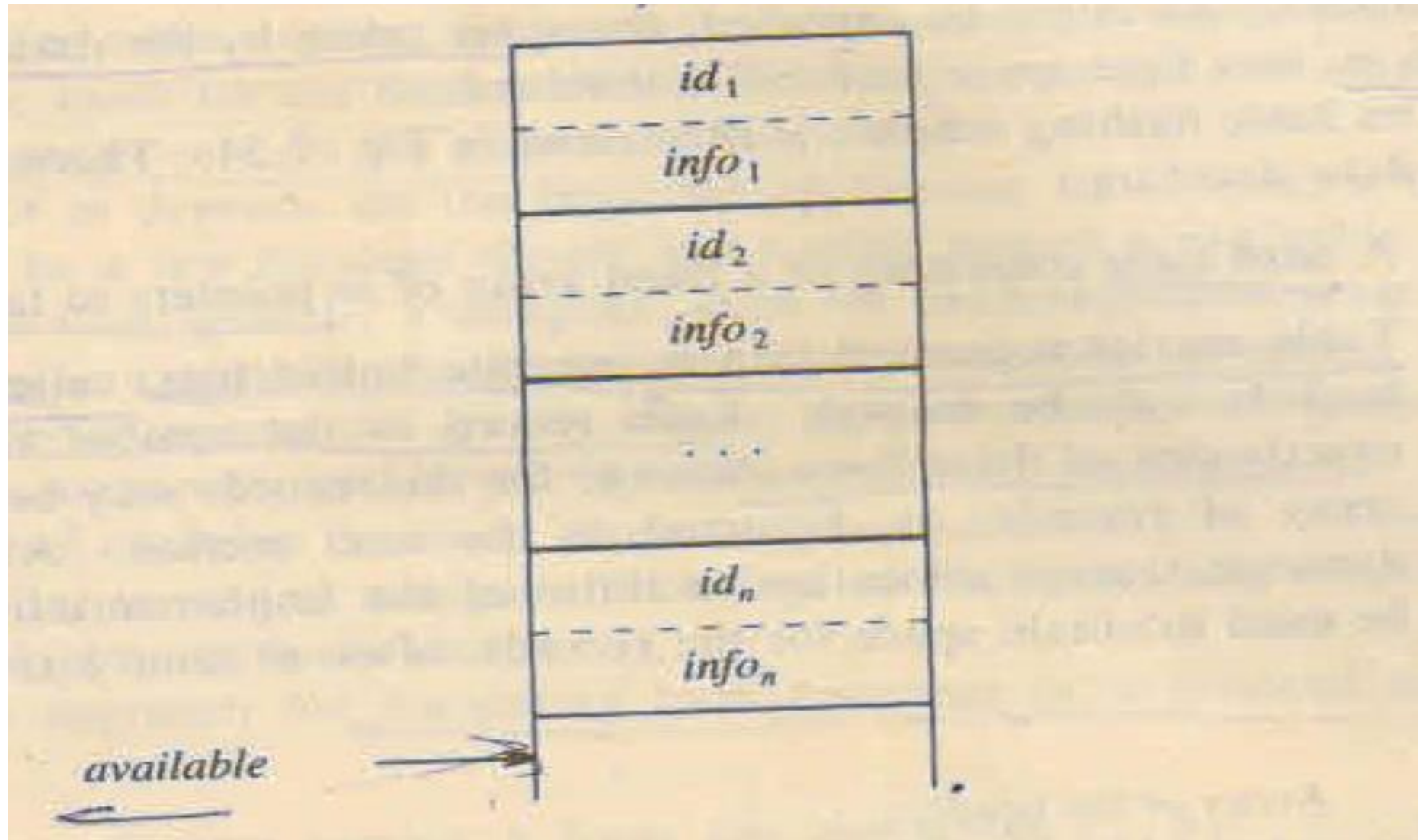
Symbol Table

In fixed-size space
within a record

NAME										ATTRIBUTES									
s	o	r	t																
a																			
r	e	a	d	a	r	r	a	y											
i																			



List Data Structure for Symbol Table



Hash Table Data Structure for Symbol Table

