Contents

Scope, Functions, and Storage Management

important topics in this chapter
parameter passing, access to global variables, storage optimization with a function call called **tail call.**

BLOCK-STRUCTURED LANGUAGES

Most modern programming languages provide some form of block.

block
is a region of program text, identified by begin and end markers, that may contain declarations local to this region.

A variable declared within a block is local to that block. A variable declared in an enclosing block is global to the block.

Storage management mechanisms associated with block structure allow functions to be called recursively.

Block-structured languages

New variables may be declared at various points in a program.

Each declaration is visible within a certain region of program text,

Blocks may be nested, but cannot partially overlap.

When a program begins executing the instructions contained in a block

memory is allocated for the variables

When a program exits a block,

the memory allocated to variables declared in that block will be deallocated.

An identifier that is not declared in the current block is considered global

local variables,
are stored on the stack in the activation record associated with the block

parameters
are also stored in the activation record associated with the block

global variables,
are declared in some enclosing block and therefore must be accessed from an activation record that was placed before activation of the current block.

Simplified Machine Model

Reference Implementation.
is an implementation of a language that is designed to define the behavior of the language.

IN-LINE BLOCKS

is a block that is not the body of a function or procedure.

Activation Records and Local Variables

When a running program enters an in-line block, space must be allocated for variables thataredeclaredintheblock.Wedothisbyallocatingasetofmemorylocationscalled **activation record** on the run-time stack.

The number of locations that need to be allocated at run time depends on the number of variables declared in the block and their types.

Intermediate Results
an activation record may also contain space for intermediate results. These are values that are not given names in the code, but that may need to be saved temporarily.

Scope and Lifetime
It is important to distinguish the scope of a declaration from the lifetime of a location:

Scope:
a region of text in which a declaration is visible.

Lifetime:
the duration, during a run of a program, during which a location is allocated as the result of a specific declaration.

Blocks and Activation Records for ML
In ML code that has sequences of declarations, we treat each declaration as a separate block.

When an ML expression contains declarations as part of the let-in-end construct, we consider the declarations to be part of the same block.

Global Variables and Control Links

operations that push and pop activation records from the run-time stack store a pointer to the top of the preceding activation record. The pointer to the top of the previous activation record is called **control link,** is the link that is followed when control returns to the instructions in the preceding block.

When a new activation record is added to the stack,

the control link of the new activation record is set to the previous value of the environment pointer, and the pointer is updated to point to the new activation record. When an activationrecordispoppedoffthestack,theenvironmentpointerisresetbyfollowing the control link from the activation record.

FUNCTIONS AND PROCEDURES

Most block-structured languages have procedures or functions that include parameters, local variables, and a body consisting of an arbitrary expression or sequence of statements.

The difference between a **procedure** and a **function** is that a function has a return value but a procedure does not. a procedure call is a statement and not an expression.

Activation Records for Functions

The **activation record** of a function or procedure block must include space for parameters and return values. a procedure may be called from different call sites, it is necessary to save the return address, **For functions,** the activation record must also contain the location that the calling routine expects to have filled with the return value of the function.

The activation record associated with a function

must contain space for the following information:

control link,

access link,

return address,

return-result address,

actual parameters of

local variables

temporary storage

Parameter Passing

The parameter names used in a function declaration are called **formal parameters.** When a function is called, expressions called **actual parameters** are used to compute the parameter values for that call.

The way that actual parameters are evaluated and passed to the function depends on the programming language The main distinctions between different parameter-passing mechanisms are

the time that the actual parameter is evaluated

the location used to store the parameter value.

Most current programming languages evaluate the actual parameters before executing the function body,

Among mechanisms that evaluate the actual parameter before executing the function body, the most common are

Pass-by-reference:

Pass-by-value:

The difference between pass-by-value and pass-by-reference is important to the programmer in several ways:

Side Effects.
Assignments inside the function body may have different effects under pass-by-value and pass-by-reference.

Aliasing.
Aliasing occurs when two names refer to the same object or location.

Efficiency.
Pass-by-value may be inefficient for large structures if the value of the large structure must be copied.

There are two ways of explaining the semantics of call-by-reference and call-byvalue. One is to draw pictures of computer memory and the run-time program stack, showing whether the stack contains a copy of the actual parameter or a reference to it. The other explanation proceeds by translating code into a language that distinguishes between Land R-values.

Semantics of Pass-by-Value

the actual parameter is evaluated. The value of the actual parameter is then stored in a new location allocated for the function parameter.

Semantics of Pass-by-Reference

the actual parameter must have an L-value. The L-value of the actual parameter is then bound to the formal parameter.

Global Variables (First-Order Case)

If an identifier **x** appears in the body of a function, but **x** is not declared inside the function, then the value of **x** depends on some declaration outside the function.

There are two main rules for finding the declaration of a global identifier:

Static Scope:

A global identifier refers to the identifier with that name that is declared in the closest enclosing scope of the program text.

Dynamic Scope:

A global identifier refers to the identifier associated with the most recent activation record.

static scope uses the static relationship between blocks in the program text. **dynamic scope** uses the actual sequence of calls that are executed in the dynamic execution of the program.

Access Links are Used to Maintain Static Scope

The **access link** of an activation record points to the activation record of the closest enclosing block in the program.

Tail Recursion (First-Order Case)

a useful compiler optimization For tail recursive functions, it is possible to reuse an activation record for a recursive call to the function. This reduces the amount of space used by a recursive function.