CSC 453, Fall 2016

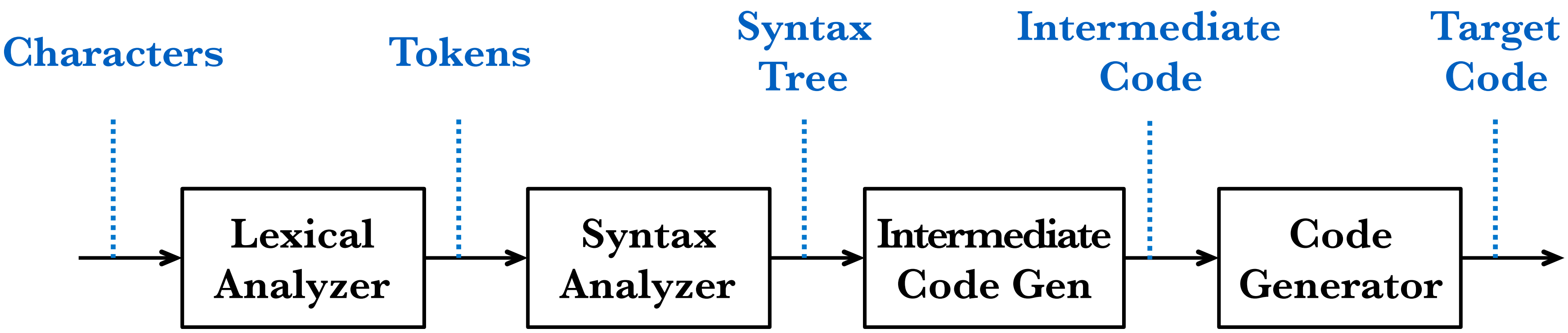
Symbol Tables

Ravi Sethi

A Sample Compiler

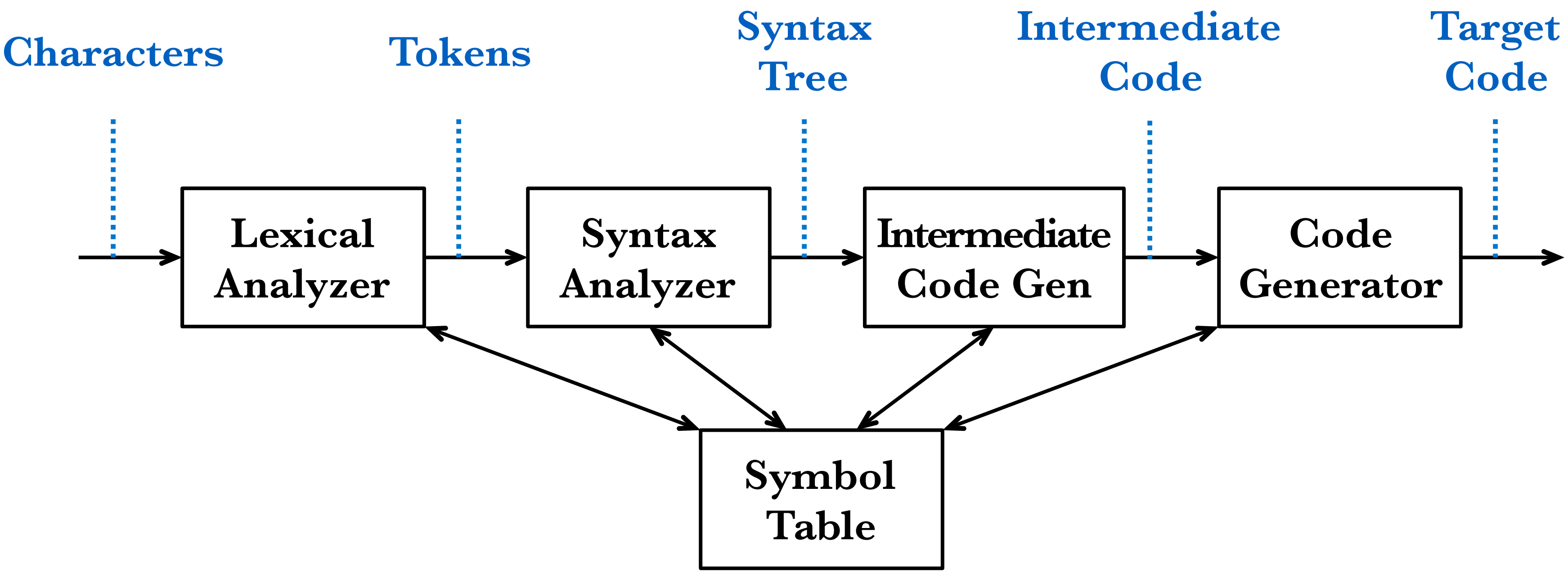
THE UNIVERSITY OF ARIZONA
Computer Science

Each phase transforms a representation of the source code



The Role of the Symbol Table

Passes information from a declaration to uses of the name



For example, type information collected incrementally during the analysis phases is used during the generation phases for storage layout.

Symbols

A symbol table associates information with names.

THE UNIVERSITY OF ARIZONA
Computer Science

"It has been remarked to me …
that once a person has understood the way in which variables are used in programming, [he or she] has understood the quintessence of programming."

— Edsger Dijkstra

Some Uses of Names

Reserve the term "identifier" for the grammar symbol

- Class names
- Variable names
- Method names
- Parameter names

Symbols

THE UNIVERSITY OF ARIZONA
Computer Science

How is **x** used in the following (from **PA4raindrop.java**)?

```
class Cloud {
    public void rain(byte x, byte y) {
        if (this.inBounds(x, y)) {
            Meggy.setPixel(x, y, Meggy.Color.BLUE);
            if (this.inBounds(x,(byte)(y+(byte)1))) {
                Meggy.setPixel(x, (byte)(y+(byte)1), Meggy.Color.DARK
            } else {}
            Meggy.delay(100);
            this.rain(x, (byte)(y-(byte)1));
        } else {}
    }
    public boolean inBounds(byte x, byte y) {
        return ((byte)(0-1) < y) && (y < (byte)8);
    }
}
```

Symbols

We'll use pseudo-code to focus on the use of names like `x`

```
class C {
    int x;
    public int f(int x) { return x; }
    public int g(int y) { return x; }
}
```

Symbols

What about `x` in the following?

```
class D {
    int x;
    public int f(int y) {
        C x = new C();
        return x.f(1);
    }
}
```

Symbols

What about `x` in the following?

```
class D {
    int x;
    public int f(int y) {
        C x = new C();
        return x.f(1);
    }
}
```

How does this pseudo-code use `f`?

Symbols

What do the occurrences of `x` denote?

```
class E {
    C x;
    public int f(int y) {
        x = new C()
        return x.x;
    }
}
```

Q. **Why would anyone write such a program?**

A. **To test a compiler.**

Scope Rules

Scope of a Declaration

THE UNIVERSITY OF ARIZONA
Computer Science

Definitions

- A **declaration** associates information with a name
- The **scope rules** of a language determine which declaration applies to an occurrence of a name
- The **scope of a declaration** is the portion of the program to which the declaration applies

Scope

THE UNIVERSITY OF ARIZONA
Computer Science

Popular usage of the term scope

- **Shorthand: scope of a name `x`**
  - Short for "scope of a the declaration of the name `x`"
- **Scope by itself**
  - A portion of a program that is the scope of one or more declarations

Static Scope Rules

Most languages have static scope rules

- **Static scope rules are based on the program text**
  - The scope of a declaration can be determined at compile time
  - Otherwise, the language is said to have dynamic scope rules
  - Macro-expansion results in dynamic scope
- **A block consists of declarations and statements**
  - Blocks are delimited by braces, `{}`, in C, Java, …
  - Blocks can be nested
  - Does MeggyJava have blocks?

Scope of a Declaration

How many declarations of x?

```
class C
{
    int x;
    public int f(int x)
    {
        return x;
    }
    public int g(int y)
    {
        return x;
    }
}
```

Scope of a Declaration

Subscripts distinguish between roles of **x**

`class C`

`{` ← Block B_1

`int` x_1`;`

`public int f(int` x_2`)`

`{` ← Block B_2

`return` x_2`;`

`}`

`public int g(int y)`

`{` ← Block B_3

`return` x_1`;`

`}`

`}`

Hole in the Scope of a Declaration

THE UNIVERSITY OF ARIZONA
Computer Science

Block B_2 is a hole in the scope of the declaration of $\mathbf{x}_1$

```
class C
    {
        int x1;
        public int f(int x2)
        {
            return x2;
        }
        public int g(int y)
        {
            return x1;
        }
    }
```

Block B_1

Block B_2

Block B_3

Most Closely Nested Rule

Find the declaration of **x** by examining blocks inside out

```
class C
    {                              ← Block B1
        int x1;
        public int f(int x2)
        {                          ← Block B2
            return x2;
        }
        public int g(int y)
        {                          ← Block B3
            return x1;
        }
    }
```

Examples of Scopes

In languages like C and Java

- **Global scope**
  - Top level declarations in C
- **Named scopes**
  - For variable and method names in a class
- **Package scopes**
  - Import a package in Java
- **Unnamed scopes**
  - Blocks

Explicit Access Control

Classes introduce a new scope for their variables and methods

- **Example:**
  - Class `C` introduces a new scope for `x`, `f`, and `g`:

```
class C {
    int x;
    public int f(...) { ... }
    public int g(...) { ... }
}
```

  - Now suppose `y` denotes an object of class `C`:

```
y = new C()
```

  - Then, `y.x` refers to variable `x` in the source text of class `C`

Explicit Access Control

The keywords `public`, `private`, `protected` control access

- `public`
  - The scope rules just discussed for classes apply without restrictions
- `private`
  - Access to the declared variable is restricted to methods of the class
- `protected`
  - Access to the declared variable is restricted to methods of the class and to the methods of any subclasses

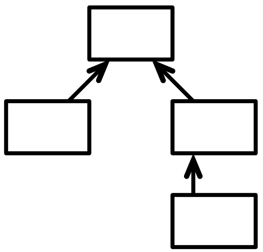
Symbol Table Per Scope

Symbol Tables

- **Kinds of information in a symbol table**
  - Type information for static checking
  - For named scopes, the identifiers in that named scope
  - Layout information for storage at run time; e.g., for storage allocation
  - …
- **Operations on symbol tables**
  - Create a new table
  - Put information in the current table
  - Get information from a chain of tables

Java Implementation of Symbol Tables

`table` is a chain of objects of class **`Env`**

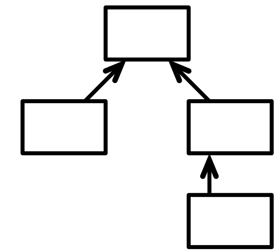


- **Creating a new table object of class `Env`**

```
public class Env {
    private hashtable table;
    protected Env previous;
    public Env(Env p) {
        table = new Hashtable() }
        previous = p;
    }
    ...
}
```

Java Implementation of Symbol Tables

`table` is a chain of objects of class `Env`



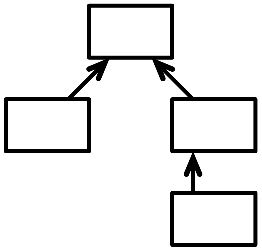
- **Get information from a chain of objects**

```
public Symbol get(String s) {
    for( Env e = this; e != null; e = e.previous ) {
        Symbol found = (Symbol)(e.table.get(s));
        if( found != null ) return found;
    }
    return null;
}
```

Handling Named Scopes

THE UNIVERSITY OF ARIZONA
Computer Science

Create a new table object for a class



- **How can we handle inheritance?**
  - Use a symbol table per class
  - The symbol table for a subclass points to the table for the superclass

Type Checking
is a form of consistency checking

Ensures that the type of a construct matches the expected type. For example,

```
Meggy.setpixel
```

expects a triple of type

$byte \times byte \times color$

Type Checking

Extending type checking from variables to expressions

- **Consider the function `inBounds`**

```
public boolean inBounds(byte x, byte y) {
    return ((byte)(0-1) < y) && (y < (byte)8);
}
```

- **It expects parameters and returns a value**
  - Parameter types (*byte*, *byte*)
  - Return value of type *boolean*

Function Signatures

THE UNIVERSITY OF ARIZONA
Computer Science

- **Consider function `f`**
  - Its parameter has type s, where s can be a tuple
  - Its return type is t
- **Then, the signature of `f` is $s \rightarrow t$**

Basic Rule of Type Checking

- If function `f` has signature $s \rightarrow t$ and `x` has type s
- Then expression `f(x)` has type t

Type Expressions

THE UNIVERSITY OF ARIZONA
Computer Science

Type checking associates type expressions with expressions

- **Basic Types**
  - *boolean*, *byte*, *int*, *color*
  - *void* denotes the absence of a value
- **Tuples**
  - If $t_1, t_2, \ldots, t_n$ are types, then $t_1 \times t_2 \times \ldots \times t_n$ is a type representing a tuple of values of types $t_1, t_2, \ldots, t_n$.
- **Functions**
  - If s and t are types, then $s \rightarrow t$ is a type expression
  - Thus, a function signature is a type expression

Type Expressions

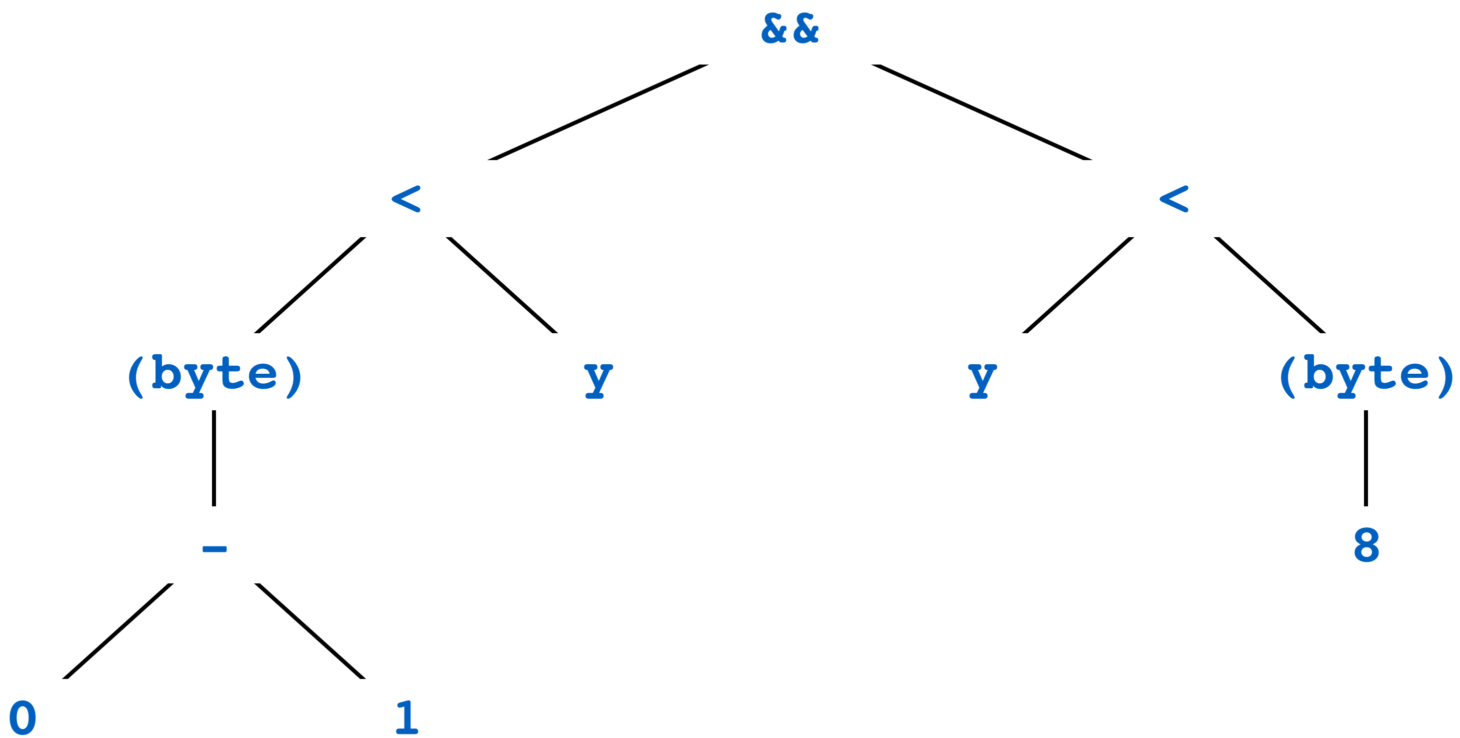
Examples: constructs and their type expressions

`8`	int
`-`	$\mathit{int} \times \mathit{int} \to \mathit{int}$
`<`	$\mathit{byte} \times \mathit{byte} \to \mathit{boolean}$
`<`	$\mathit{int} \times \mathit{int} \to \mathit{boolean}$
`&&`	$\mathit{boolean} \times \mathit{boolean} \to \mathit{boolean}$
`(byte)`	$\mathit{int} \to \mathit{byte}$

A function with more than one signature is said to be overloaded.

An Expression Tree

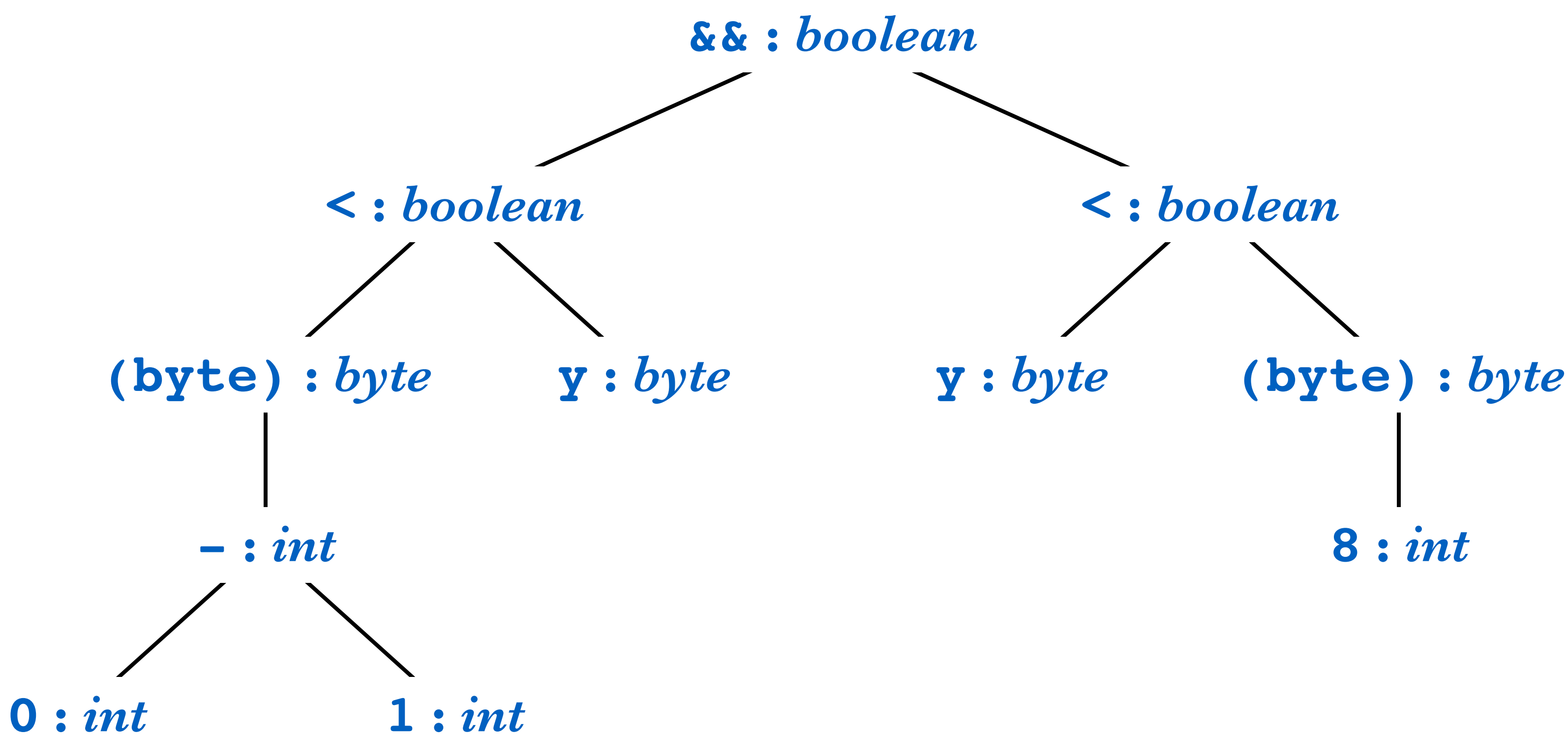
Expression `((byte)(0-1) < y) && (y < (byte)8)`



Type Checking

THE UNIVERSITY OF ARIZONA
Computer Science

Associate a type expression with each subexpression



Type Expressions for Statement Nodes

THE UNIVERSITY OF ARIZONA
Computer Science

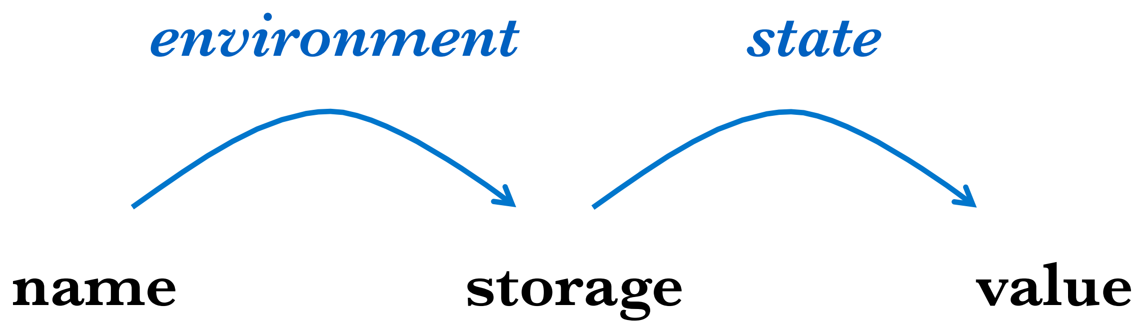
Allows uniform treatment of nodes in a syntax tree

- **Treat `while` as a function with signature**
  - $boolean \times void \rightarrow void$
- **Similar treatment for other statement nodes**

Lifetime

A consecutive sequence of steps at run time

Two-Stage Mapping of Names to Values



- **The lifetime of a declaration**
  - The consecutive sequence of steps during which the declared name has
    storage and a value
  - In other words, the state mapping is defined
- **Lifetime does not equate to accessibility**
  - Example: a nested block may have another declaration of the name
  - In other words, the environment may change

Scope and Lifetime of a Declaration

THE UNIVERSITY OF ARIZONA
Computer Science

The value of $\mathbf{x}_1$ is inaccessible during the lifetime of $\mathbf{x}_2$

```
class C
    {
        int x1;
        public int f(int x2)
        {
            return x2;
        }
        public int g(int y)
        {
            return x1;
        }
    }
```

Block B_1 (the class body `{ … }`)

Block B_2 (the body of `f`)

Block B_3 (the body of `g`)

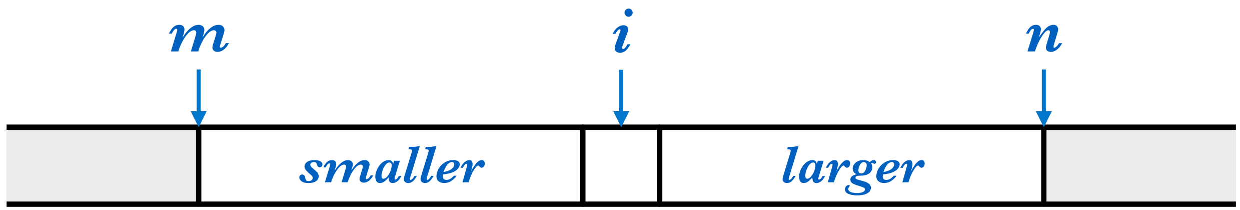
Activation Trees

Handling of local variables in recursive activations

A recursive function

THE UNIVERSITY OF ARIZONA
Computer Science

Function *quicksort* has parameters m and n and a local var i



- **To sort array elements in the range m:n**
  - Pick a pivot element i
  - Partition the elements into two groups: smaller and larger than the pivot
  - Recursively *quicksort* the ranges m:i–1 and i+1:n
  - Sort the whole array by calling *quicksort* with the lower and upper bounds of the array

Activations

- **An activation of a function is an execution of the function body**

- **Activations can be nested**
  - **If an activation of *f* initiates an activation of *g*, then that activation of *g* is nested in that activation of *f***

Trace from an activation of quicksort

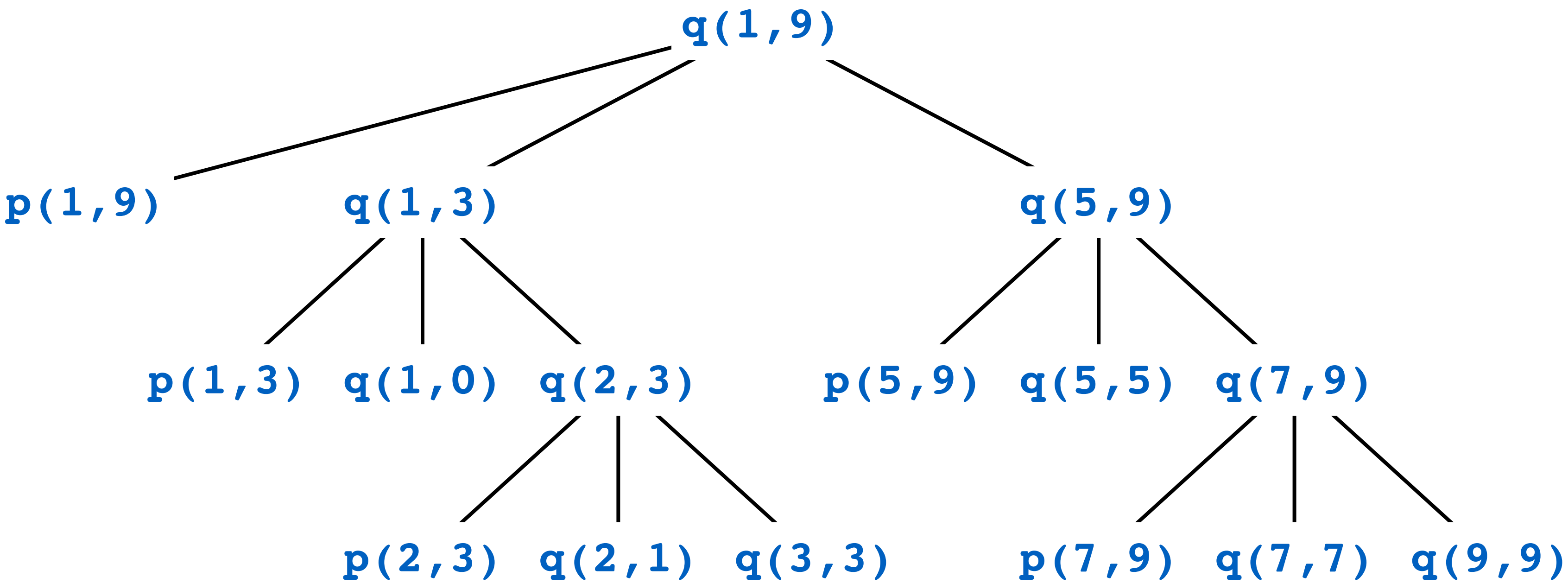
THE UNIVERSITY OF ARIZONA
Computer Science

Parameters are in parentheses

```
enter quicksort(1,9)
    enter partition(1,9)
    leave partition(1,9)
    enter quicksort(1,3)
        ...
    leave quicksort(1,3)
    enter quicksort(5,9)
        ...
    leave quicksort(5,9)
leave quicksort(1,9)
```

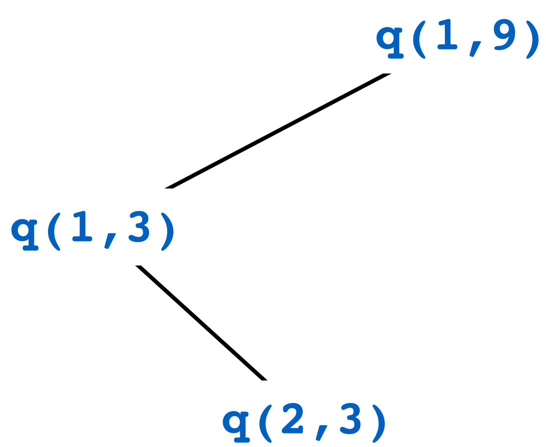
Activation Tree

Abbreviations: `q` for `quicksort`, `p` for `partition`



Live Activations are Nested

Live activations when control reaches `q(2,3)`



- **We can use a stack to keep track of live activations**
  - Called a run-time stack
- **What does a local variable *i* in *q* denote?**
  - What is its scope?
  - What is its lifetime?

Symbol Tables

The University of Arizona
Computer Science

Key Points

- **Static scope rules can be applied at compile time**
  - We deal with the scope of a declaration of a name in the source text
- **Symbol table per scope**
  - Holds information that a declaration associates with a name
  - Information collected in one phase can be used in another
- **Type Checking**
  - Associate a type expression with nodes in a syntax tree
- **Lifetime is a run-time concept**
  - We deal with the lifetime of an activation of a local variable