Chapter 2 Section 4
Abstract Computation Background
Introduction to Programming Languages
First Edition, 2013

Author: Arvind Bansal

ISBN: 978-146-6565142

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 1

Topics covered

- Recursion vs. Iteration
- Sequence
- Data and Reference
- Recursive data structures
- Abstract concept in computation
- Summary

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 2

Recursion

- **A definition uses itself to define with different argument**
  - At least one base case and at least one recursive definition
  - Progressively unfolds and moves towards the base case
  - Previous invocations are suspended until next recursive invocation returns value
  - Number of invocations decided by the input value
- **Example: factorial function or fibonacci function**

  factorial(0) = 1. % base clause
  factorial(n) = n * factorial(n – 1) % recursive definition

  fibonacci(0) = 1.
  fibonacci(1) = 1.
  fibonacci(n) = fibonacci(n – 1) = fibonacci(n – 2).

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 3

Recursion vs. Iteration

- Implementation of recursion
  - A stack is needed to hold the execution space needed by variables for every procedure invocation
  - Stack has memory and execution overhead of calling and returning from called recursive procedures.

Iteration	Recursion
No overhead of calling and returning from called procedure	Excessive overhead of calling and returning from recursive invocation
starts from the base case, and reuses the memory locations to accumulate results	Suspends recursive calls that needs additional memory before hitting the base case
more efficient execution	Slow due to overheads

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 4

Optimizing Recursive programs

- **Tail recursion**
  - Recursive call is the last one in the definition.
  - Is equivalent to indefinite iteration.
  - Tail recursion can be transformed to equivalent indefinite iteration
- **Linear recursive programs**
  - Has only one recursive call to itself in the recursive definition
  - Can be transformed to indefinite indefinite iteration

```
Algorithm iterative_factorial
Input: input  value n;
Output: accumulator  value;
{     acc = 1;
      for (i = 1; i =< n; i++)
          acc  = i *   acc;
}
```

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 5
Author: Arvind Bansal

Sequence

- **Definition: a bag to model ordered collection of entities**
- **Representation: modeled within angular brackets**
  - <a, b, c>
- **Operations**
  - Finding an element by position
  - Insertion and deletion of elements by index
  - First, second, last elements of a sequence
  - Deletion and substitution of a subsequence by content
  - Joining two sequences
  - Finding out predecessor and successor of an entity in a sequence
- **Application**
  - multiple data types can be modeled as sequence such as stacks, queues, files, strings etc.

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 6
Author: Arvind Bansal

Data and Reference

- **Memory locations hold two types of information**
  - Data and reference to memory locations
  - Pointers are addresses of memory location stored in another memory location or processor registers
- **Advantages of pointers**
  - Minimal overhead of data movement
  - Supports recursive data structures (lists, trees) and dynamic objects
  - Delaying memory allocation of variables until runtime
  - Allocating physically separated chained memory blocks for logically contiguous data structures
  - Sharing memory blocks among multiple data structures
  - Providing independence of the program from data movement
- **Disadvantages of pointers**
  - Arithmetic operations on pointers causes segment hopping error.
  - Shared blocks can not be reused until all pointers are released.

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 7
Author: Arvind Bansal

Recursive Data Structures

- **Definition: A data structure that uses itself in the definition**
  - One or more recursive definition and one or more base definition
- **Examples: linked list, tree, vector**
- **Linked list**
  <linked-list> ::= *<data-element>* *<linked-list>* | null
- Trees
  <binary-tree> ::= *<binary-tree>* *<data-element>* *<binary-tree>* | *void*
- **Implementation**
  - Uses pointers or references to implement.
  - Pointers are used so that memory allocation can be done at runtime as needed.

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 8
Author: Arvind Bansal

Abstract Computation - I

- **Program is a sequence of meaningful instructions (statement)**
- **Each statement is terminated by a delimiter or linefeed**
- **A program can have**
  - Literals, l-values, r-values, identifiers, labels, definitions, declarations, assignment statement, commands, expressions, procedures and functions, strings, procedure invocations, parameters, and sequencers
  - **Literal –** an elementary expression that can not be further split. Examples: number, character, atom
  - **r-value** – the evaluated value of an expression. Occurs on the right hand side of an assignment. Actual value of a variable
  - **l-value** – location value of a variable. Occurs on the left hand side of an assignment
  - **Identifier** – a symbolic name associated with an entity such as constant, procedure, variable etc.
  - **Definition** – A symbol associated with a value. During compilation symbol is substituted by the corresponding value

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 9

Abstract Computation II

- **Variable**
  - identifier → l-value → r-value
  - can be associated with a concrete value or type
  - associated with a type is called type variable
  - can be destructively updated or could be assign-once
- Assignment statement
  - Right hand side expression is evaluated and written into the memory location associated with the variable name.
- **Command is a statement with embedded assignment statement**
- **Expression evaluation does not write into memory location**

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 10

Abstraction Computation III

- **String is a sequence of characters**
- **Operators could be**
  - Dyadic – having two operands such as addition, subtraction, multiplication, division, logical or, logical and
  - Monadic – having one operand such as not, - *<operand>*
- **Mutable vs. Assign-once variables**

Mutable	Assign-once
1. Reusable 2. Looses past information 3. Undesired program-behaviors due to side-effects	1. Memory explosion 2. Use of past values to find alternate solutions 3. Does not support iteration 4. Less side-effects

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 11

Binding and Scope Rules

- Binding
  - An entity is associated with corresponding attributes
- **<u>Example</u>**
  - Variable-name bound to a memory location
  - Memory location bound to an r-value
  - Identifier bound to a procedure-block
- **Scope Rule**
  - Defines the visibility of declarations within a part of the programs
  - Can be static or dynamic
  - Static binding means visibility does not change with program execution
  - Dynamic binding means visibility changes with procedure invocation, and unbound variables pick up the value from the declarations in the reverse order of the invoked procedures.

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 12

Examples of Scope Rules

Static scope rule

```
main ( )
{   integer x, y, z;
    x = 4; y = 10; z = 12;
    {integer temp, z;
       temp = x; x = y; y = temp; z = 5;}
    print( x, y, z);
}
```

- outer block: x, y, z
- inner-block: temp, z-inner, x, y
- Z-outer is shadowed in the inner block

Dynamic scope rule

```
integer sum(integer x);
return (x + y);
main ( )
{ {integer y, z; y = 4; z = 5; sum(y);}
   {integer w, y, z;
       w = 4, y = 5; z = 6; sum(z);}
}
```

- first call to sum(y) returns 8; In sum, x gets bound to value of y = 4, and y gets bound to 4.
- Second call sum(z) returns 11. In sum, x gets bound to z = 6, and y gets bound to 5

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 13
Author: Arvind Bansal

Types of Variables

- **Variables are classified by visibility rules and lifetime**
  - Global – visible everywhere, lifetime throughout the program
  - Nonlocal – visible in the nested procedures, lifetime is the procedure in which it is declared
  - Local – visible within the procedure they are declared
- **Variables can be static or dynamic**
  - Static variables are allocate memory location at compile time
  - Dynamic variables are allocated memory locations during runtime
- **Variables in object-oriented languages**
  - Class variables – variables declared in class, accesses by all instance of the class
  - Instance variable – only accessible in a specific object

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 14
Author: Arvind Bansal

Environment and Store

Environment

- **Environment is set of mapping between identifier and memory locations**
- **Environment changes with a new declaration**
  - Creating new identifier → memory location mapping
  - Shadowing non local variable

Store

- **Store is mapping of memory location to r-value**
- **Store changes with a new assignment statement or initialization or parameter passing**

Global variables → l-value

Non-local variables → l-value

Local variables → l-value

Reference parameters
Formal parameters → l-value

Environment

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 15
Author: Arvind Bansal

Functions and Procedures

Function

- **Function is a collection of expressions**
- **Function does not alter the store as it has no destructive updates**
- **Functions has four components**
  - Name, body, parameters and bounded variables

Procedure

- **Procedure contains atleast one command**
- **Procedure alters the store due to assignment statements**

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 16
Author: Arvind Bansal

Abstracting the Program Execution

- **Program execution is modeled as a state transition**
  - Each statement transforms program to a new state.
  - A state is a triple of the form (σ^E, σ^S, σ^D) where σ^E is the environment, σ^S is the store, and σ^D is a stack of pairs of the form (σ^E, σ^S) of the suspended calling procedures in LIFO order.
  - Computational state changes when environment changes, when store changes, when a procedure is called, and when control returns from a procedure
- **Program execution modeled as Boolean state transition**
  - Each state is a Boolean conditions connected through logical operators: logical-and, logical-or and negation
  - Boolean expression changes each time an assignment operation is executed
  - Example: X = 5 + 3 makes X == 8 as true

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 17

Summary

- Abstract computation concepts are used to model the program execution abstractly
  - There are many abstract entities such as literal, l-value, r-value, variables, definitions, assignment, expression, command
  - Variable is identifier → l-value → r-value
  - Environment is a set of mapping of identifier → memory locations
  - Store is a set f mapping of memory locations → values
  - An assignment statement destructively updates a memory location
  - A command contains at least one assignment statement
  - An expression does not have an assignment statement
- Scope could be static or dynamic
  - Static scope rule is based upon program structure
  - Dynamic scope rule is based upon the LIFO pattern of the calling procedures

Introduction to Programming Languages, 1st edition, 2013, **ISBN**: 978-146-6565142 Slide 18
Author: Arvind Bansal © Chapman Hall/CRC Press, 2013, All rights reserved