Unit 7

Primitives, pseudocode, problem solving, decisions, loop control

Pretest loop: repetition structure that looks before it leaps

Stepwise refinement: divide and conquer approach to problem solving

Primitive: basic building block of programming languages

Pseudocode: informal notation for representing algorithms

Algorithm: the fundamental concept in computer science

If-then-else statement: means of producing different actions depending on a condition

Procedure: program segment isolated as a unit with a name

Assignment statement: means of saving the result of a computation for future use

Which of the code examples defines an algorithm in the strict sense?

X=3

While x < 5:

X=x+1

Which of the following is NOT a means of repeating a block of instructions?

- assignment

- not pretest loop, posttest loop, recursion

Indentation and comments enhance the readability of a computer program

Stepwise refinement is not a way of representing algorithms

While is a python primitive that corresponds to 2 JUMP instructions

A process is the activity of executing a program, which a program is a representation of an algorithm

Once a function is correctly constructed, it can be used as a building block for larger program structures without reconsidering the functions internal composition

5.1-5.4

Definition of Algorithm

* An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process.

Algorithm Representation

* Requires well-defined primitives
* A collection of primitives constitutes a programming language.

Figure 5.2 Folding a bird from a square piece of paper

Figure 5.3 Origami primitives

Pseudocode Primitives

* Assignment

*name* = *expression*

* Example

RemainingFunds = CheckingBalance +

SavingsBalance

* Conditional selection

if (*condition*):

*activity*

* Example

if (sales have decreased):

lower the price by 5%

* Conditional selection

if (*condition*):

*activity*

else:

*activity*

* Example

if (year is leap year):

daily total = total / 366

else:

daily total = total / 365

* Repeated execution

while (*condition*):

*body*

* Example

while (tickets remain to be sold):

sell a ticket

* Indentation shows nested conditions

if (not raining):

if (temperature == hot):

go swimming

else:

play golf

else:

watch television

* Define a function

def *name*():

* Example

def ProcessLoan():

* Executing a function

if (. . .):

ProcessLoan()

else:

RejectApplication()

Figure 5.4 The procedure Greetings in pseudocode

* Using parameters

def Sort(List):

.

.

* Executing Sort on different lists

Sort(the membership list)

Sort(the wedding guest list)

Polya’s Problem Solving Steps

1. Understand the problem.

2. Devise a plan for solving the problem.

3. Carry out the plan.

4. Evaluate the solution for accuracy and its potential as a tool for solving other problems.

Polya’s Steps in the Context of Program Development

1. Understand the problem.

2. Get an idea of how an algorithmic function might solve the problem.

3. Formulate the algorithm and represent it as a program.

4. Evaluate the solution for accuracy and its potential as a tool for solving other problems.

Getting a Foot in the Door

Try working the problem backwards

Solve an easier related problem

Relax some of the problem constraints

Solve pieces of the problem first (bottom up methodology)

Stepwise refinement: Divide the problem into smaller problems (top-down methodology)

Ages of Children Problem

Person A is charged with the task of determining the ages of B’s three children.

B tells A that the product of the children’s ages is 36.

A replies that another clue is required.

B tells A the sum of the children’s ages.

A replies that another clue is needed.

B tells A that the oldest child plays the piano.

A tells B the ages of the three children.

How old are the three children?

Figure 5.5

Figure 5.6 The sequential search algorithm in pseudocode

Figure 5.7 Components of repetitive control

Initialize: establish an initial state that will be modified toward the termination condition

Test: compare the current state to the termination condition and terminate the repetition if equal

Modify: change the state in such a way that it moves toward the termination condition

Iterative Structures

Pretest loop:

while (*condition*):

*body*

Posttest loop:

repeat:

*body*

until(*condition*)

Figure 5.8 The while loop structure

Figure 5.9 The repeat loop structure

Figure 5.10 Sorting the list Fred, Alex, Diana, Byron, and Carol alphabetically

Figure 5.11 The insertion sort algorithm expressed in pseudocode

def Sort(List):

N = 2

while (N <= length of List):

Pivot = Nth entry in List

Remove Nth entry leaving a hole in List

while (there is an Entry above the

hole and Entry > Pivot):

Move Entry down into the hole leaving

a hole in the list above the Entry

Move Pivot into the hole

N = N + 1

Recursion

* The execution of a procedure leads to another execution of the procedure.
* Multiple activations of the procedure are formed, all but one of which are waiting for other activations to complete.

Figure 5.12 Applying our strategy to search a list for the entry John

Figure 5.13 A first draft of the binary search technique

if (List is empty):

Report that the search failed

else:

TestEntry = middle entry in the List

if (TargetValue == TestEntry):

Report that the search succeeded

if (TargetValue < TestEntry):

Search the portion of List preceding TestEntry for

TargetValue, and report the result of that search

if (TargetValue > TestEntry):

Search the portion of List following TestEntry for

TargetValue, and report the result of that search

Figure 5.14 The binary search algorithm in pseudocode

def Search(List, TargetValue):

if (List is empty):

Report that the search failed

else:

TestEntry = middle entry in the List

if (TargetValue == TestEntry):

Report that the search succeeded

if (TargetValue < TestEntry):

Sublist = portion of List preceding TestEntry

Search(Sublist, TargetValue)

if (TargetValue < TestEntry):

Sublist = portion of List following TestEntry

Search(Sublist, TargetValue)

Figure 5.15

Figure 5.16

Figure 5.17

Algorithm Efficiency

* Measured as number of instructions executed
* Big theta notation: Used to represent efficiency classes
* Example: Insertion sort is in Θ(n2)
* Best, worst, and average case analysis

Figure 5.18 Applying the insertion sort in a worst-case situation

Figure 5.19 Graph of the worst-case analysis of the insertion sort algorithm

Figure 5.20 Graph of the worst-case analysis of the binary search algorithm

Software Verification

* Proof of correctness
* Assertions
* Preconditions
* Loop invariants
* Testing

Chain Separating Problem

* A traveler has a gold chain of seven links.
* He must stay at an isolated hotel for seven nights.
* The rent each night consists of one link from the chain.
* What is the fewest number of links that must be cut so that the traveler can pay the hotel one link of the chain each morning without paying for lodging in advance?

Figure 5.21 Separating the chain using only three cuts

Figure 5.22 Solving the problem with only one cut

Figure 5.23 The assertions associated with a typical while structure

Unit 8

paradigms, compiler vs interpreter, variables, functions, scope

source code: programs written in a high-level language and stored in a plain text file

method: object-oriented terminology for procedure or function

compiler: program that translates source code into an executable file

operator precedence: dictates the order in which mathematical or other operations are performed

programming paradigm: approach to the programming process that influences the design of a language

parameter: means of passing info to a procedure or function

interpreter: program that executes source code directly without creating an executable file

structured programming: design principles that encourage appropriate use of control statements

most machine languages are based on the imperative paradigm

assignment doesn’t require a Boolean structure

comment statements are ignored by a compiler

type is not a way of referring to a value in a program

scope of a variable is the portion of the code where the variable exists

in python, import makes it possible to reuse code from other files

in the context of the object-oriented paradigm, classes are templates from which objects are constructed. We say that the latter is an instance of the former

a global variable is readily accessible throughout the program, whereas a local variable is accessible only within a specific are

print hello, if value <0, while not done:, count = count + 1 are imperative

def sleep(seconds):, class Finch: are declarative

6.1-6.3

Chapter 6: Programming Languages

* 1. Historical Perspective
  2. Traditional Programming Concepts
  3. Procedural Units

A program in a third-generation language is machine independent in the sense that its steps are not stated in terms of the machine’s attributes suc h as registers and memory cell addresses. On the other hand, it is machine dependent in the sense that arithmetric overflow and truncation errors will still occur.

The major distnict is that an assembler translates each instruction in th source program into a single machine instruction, whereas a comiler often produces many machine-language instructions to obtain the equivalent of a single source program instruction.

The declarative paradigm is based on developing a description of the problem to be solved. The functional paradigm forces the programmer to describe the problem’s solution in terms of solutions to smaller problems. The object-oriented paradigm places emphasis on describing the components in the problem’s environment.

The

Fruitful function is a function that returns a value associated with the function’s name.

Figure 6.1 Generations of programming languages

Second-generation: Assembly language

* A mnemonic system for representing machine instructions
  + Mnemonic names for op-codes
  + Program variables or identifiers: Descriptive names for memory locations, chosen by the programmer

Assembly Language Characteristics

* One-to-one correspondence between machine instructions and assembly instructions
  + Programmer must think like the machine
* Inherently machine-dependent
* Converted to machine language by a program called an assembler

Program Example

Machine language  
156C  
166D  
5056  
30CE  
C000

Assembly language  
LD R5, Price  
LD R6, ShipCharge  
ADDI R0, R5 R6  
ST R0, TotalCost  
HLT

Third generation language

* Uses high-level primitives
  + Similar to our pseudocode in Chapter 5
* Machine independent (mostly)
* Examples: FORTRAN, COBOL
* Each primitive corresponds to a sequence of machine language instructions
* Converted to machine language by a program called a compiler

Figure 6.2 The evolution of programming paradigms

Figure 6.3 A function for checkbook balancing constructed from simpler functions

Figure 6.4 The composition of a typical imperative program or program unit

* The first part consists of declaration statements describing the data that is manipulated by the program
* The second part consists of imperative statements describing the action to be performed

Data Types

* Integer: Whole numbers
* Real (float): Numbers with fractions
* Character: Symbols
* Boolean: True/false

Variables and Data Types

float Length, Width;

int Price, Total, Tax;

char Symbol;

int WeightLimit = 100;

Data Structure

* Conceptual shape or arrangement of data
* A common data structure is the array
* In C

int Scores[2][9];

* In FORTRAN

INTEGER Scores(2,9)

* Figure 6.5 A two-dimensional array with two rows and nine columns
* Scores (2,4) in FORTRAN where indices start at one
* Scores (1) (3) in C and its derivatives where indices start at zero
* Figure 6.6 The conceptual structure of the aggregate type Employee

Assignment Statements

* In C, C++, C#, Java

Z = X + y;

* In Ada

Z := X + y;

* In APL (A Programming Language)

Z ← X + y

Control Statements

* Go to statement

goto 40

20 Evade()

goto 70

40 if (KryptoniteLevel < LethalDose) then goto 60

goto 20

60 RescueDamsel()

70 ...

* As a single statement

if (KryptoniteLevel < LethalDose):

RescueDamsel()

else:

Evade()

* If in Python

if (condition):

statementA

else:

statementB

* In C, C++, C#, and Java

if (condition) statementA; else statementB;

* In Ada

IF condition THEN

statementA;

ELSE

statementB;

END IF;

* While in Python

while (condition):

body

* In C, C++, C#, and Java

while (condition)

{ body }

* In Ada

WHILE condition LOOP

body

END LOOP;

* Switch statement in C, C++, C#, and Java

switch (variable) {

case 'A': statementA; break;

case 'B': statementB; break;

case 'C': statementC; break;

default: statementD; }

* In Ada

CASE variable IS

WHEN 'A'=> statementA;

WHEN 'B'=> statementB;

WHEN 'C'=> statementC;

WHEN OTHERS=> statementD;

END CASE;

* Figure 6.7 The for loop structure and its representation in C++, C#, and Java

Comments

* Explanatory statements within a program
* Helpful when a human reads a program
* Ignored by the compiler

/\* This is a comment. \*/

// This is a comment

Procedural Units

* Many terms for this concept:
  + Subprogram, subroutine, procedure, method, function
* Unit begins with the function’s header
* Local versus Global Variables
* Formal versus Actual Parameters
* Passing parameters by value versus reference
* Figure 6.8 The flow of control involving a function
* Figure 6.9 The function ProjectPopulation written in the programming language C
* Figure 6.10 Executing the function Demo and passing parameters by value
* Figure 6.11 Executing the function Demo and passing parameters by reference
* Figure 6.12 The fruitful function CylinderVolume written in the programming language C
* Figure 6.13 The translation process
* Figure 6.14 A syntax diagram   
  of Python’s if-then-else statement
* Figure 6.15 Syntax diagrams describing the structure of a simple algebraic expression
* Figure 6.16 The parse tree for the string x + y \* z based on the syntax diagrams in Figure 6.17
* Figure 6.17 Two distinct parse trees for the statement if B1 then if B2 then S1 else S2
* Figure 6.18 An object-oriented approach to the translation process

Objects and Classes

* Object: Active program unit containing both data and procedures
* Class: A template from which objects are constructed

An object is called an instance of the class.

* Figure 6.19 The structure of a class describing a laser weapon in a computer game

Components of an Object

* Instance Variable: Variable within an object
  + Holds information within the object
* Method: Procedure within an object
  + Describes the actions that the object can perform
* Constructor: Special method used to initialize a new object when it is first constructed
* Figure 6.21 A class with a constructor
* Object Integrity
* Encapsulation: A way of restricting access to the internal components of an object
  + Private
  + Public
* Figure 6.22 Our LaserClass definition using encapsulation as it would appear in a Java or C# program

Additional Object-oriented Concepts

* Inheritance: Allows new classes to be defined in terms of previously defined classes
* Polymorphism: Allows method calls to be interpreted by the object that receives the call
  + Programming Concurrent Activities
* Parallel (or concurrent) processing: simultaneous execution of multiple processes
  + True concurrent processing requires multiple CPUs
  + Can be simulated using time-sharing with a single CPU
* Figure 6.23 Spawning threads

Controlling Access to Data

* Mutual Exclusion: A method for ensuring that data can be accessed by only one process at a time
* Monitor: A data item augmented with the ability to control access to itself

Declarative Programming

* Resolution: Combining two or more statements to produce a new statement (that is a logical consequence of the originals).
  + Example: (P OR Q) AND (R OR ¬Q)   
    resolves to (P OR R)
  + Resolvent: A new statement deduced by resolution
  + Clause form: A statement whose elementary components are connected by the Boolean operation OR
* Unification: Assigning a value to a variable so that two statements become “compatible.”
* Figure 6.24 Resolving the statements (*P* OR *Q*) and (*R* OR ¬*Q*) to produce (*P* OR *R*)
* Figure 6.25 Resolving the statements (*P* OR *Q*), (*R* OR ¬*Q*), ¬R, and ¬*P*

Prolog

* Fact: A Prolog statement establishing a fact
  + Consists of a single predicate
  + Form: *predicateName*(*arguments*).
    - Example: parent(bill, mary).
* Rule: A Prolog statement establishing a general rule
  + Form: *conclusion* :- *premise.*
    - :- means “if”
  + Example: wise(X) :- old(X).
  + Example: faster(X,Z) :- faster(X,Y), faster(Y,Z).