2. Design

**Topic: Program design**
- The software development process
- Pseudocode and flowcharts
- Control structures:
  - Sequence
  - Decision
  - Loop
- Modular decomposition and stepwise refinement
- Object-oriented design and UML

**The software development process**
- Understanding the problem: *what*
  - input
  - output
  - their relationship
- Designing a solution: *how*
- Algorithms are language independent
- Coding, testing come later
- Development process is iterative

**Two notations for low-level design**
- Both notations show order of execution
- Pseudocode
  - informal
  - precise
  - text outline format
- Flowcharts
  - graphical
  - shapes denote steps
  - arrows show flow of control

**Algorithm:**
A precise plan to solve a problem in a finite number of steps
- Program designs use algorithms
- Most computation is algorithmic
- Flowcharts and pseudocode can express algorithms

**The sequence control structure**
*Problem:*
Accomplish dinner routine

*Pseudocode:*
1. Cook dinner
2. Eat dinner
3. Clean up

*Flowchart:*

**An algorithm to add numbers**
1. Prompt for integers *input1, input2*
2. *sum ← input1 + input2*
3. Display *sum*

- In pseudocode and flowcharts, the symbol ← stands for assignment of a value to a variable
- An algorithm is almost always *general purpose*: it works with *variables*, not only specific constant values
2. Design

The decision control structure

Problem:
Prepare to do homework

Pseudocode:
If exercises are confusing
review chapter

Flowchart:

Finding absolute value

Input n
If \( n \geq 0 \)
display n
otherwise
display \((-n)\)

• With the branch control structure,
one and only one of the alternatives
executes
• In pseudocode, the subordinate (conditional) steps are normally indented

Multiple alternatives

• Require multiple diamonds in
flowchart

If \( n = 1 \) display ‘a’
otherwise
If \( n = 2 \) display ‘b’
otherwise
If \( n = 3 \) display ‘c’
otherwise
If \( n = 4 \) display ‘d’

The repetition (loop) control structure

Problem:
Telephone someone

Pseudocode:
Repeat
  Dial number
until someone answers

Counting to 10

• A counted loop; counter is \( i \)

Loop invariants

• A loop invariant is a logical (true/false)
assertion about values and outputs at the
beginning of a loop iteration
• Example
  Loop invariants for the previous slide are
  – The value of \( i \) is between 0 and 11
  – All values in \{1 ... \( i - 1 \)\} have been displayed
• Loop invariants help assure correctness
### Tracing an algorithm

- Allows designer to check result of algorithm, including internal (undisplayed) values
- Use one column per value traced; one row per loop iteration.
- Example (See previous slide):

<table>
<thead>
<tr>
<th>quantity</th>
<th>total</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

### Three control structures are sufficient

- Three control structures: sequence, branch, and loop
- They are used in structured programming; may be combined
- Any computable problem can be solved using only these three
- Note: Some problems are not computable

### Combining, nesting control structures

- Loop within loop
- Branch within loop

### Structured flowcharts have one entrance, one exit

- Structured pseudocode also avoids “Go to step…”

### The picture tells a story

- A flowchart should show what happens, at a glance
2. Design

Modular decomposition
- One strategy: divide and conquer
- All languages use modularity
- An organization is modular
- Modular design may be top-down
- Subprograms implement modular designs

Module hierarchy charts tell which subprogram uses which

- Module do_problem uses modules calculate, get_input and display_result
- calculate uses add and divide
- Stepwise refinement may lead to further decomposition
- Module hierarchy contains names, not steps, in contrast to flowchart

A divide-and-conquer strategy to solve problem of walking \( n \) steps

**Walk (steps)**

If steps > 0
- Take a step
- Walk (steps – 1)

- This algorithm is recursive because it uses itself

Review of functions
- **Function**: A set that is a mapping from one set to itself or to another set
- **Examples**:
  - Index('B') = 2
  - Odd(3) = true
  - Twice(1) = 2

Operators and functions
- Two possible ways to express the same transformation of data
- **Examples**:
  - \( a + b \) \( \text{sum}(a,b) \)
  - \( (2 + 5) \times 4 \) \( \text{product}(\text{sum}(2,5), 4) \)
  - \( \sum_{k=1}^{4} k \) \( \text{sum}(1,2,3,4) \)

Computable function:
- A mapping obtained by an effective step-by-step procedure
- **Examples**: successor, sum
- Computer programming is concerned with coding computable functions
- An uncomputable function: “Does this program work correctly?”
2. Design

Functions vs. computation
- A function in mathematics is a passive set of pairs
- A computation entails action
- A computer program or subprogram computes a mathematical function
- A C or C++ “function” is actually a subprogram, not a mathematical function

Recursive functions and computable functions
- For natural numbers \( a, b \) in \( \{0, 1, 2, 3\ldots\} \):
  \[
  \text{sum}(a, b) = \begin{cases} 
  b & \text{if } a = 0 \\
  \text{sum}(a-1, b+1) & \text{otherwise}
  \end{cases}
  \]
- The definition of \( \text{sum} \) is recursive because it uses itself
- The form of this definition is a recurrence
- It guarantees computability

Some recurrences
\[
\begin{align*}
\text{sum}(a, b) &= \begin{cases} 
  b & \text{if } a = 0 \\
  \text{sum}(a-1, b+1) & \text{otherwise}
  \end{cases} \\
\text{product}(a, b) &= \begin{cases} 
  0 & \text{if } b = 0 \\
  a & \text{if } b = 1 \\
  a + \text{product}(a, b-1) & \text{otherwise}
  \end{cases} \\
\text{even}(a) &= \begin{cases} 
  \text{true} & \text{if } a = 0 \\
  \text{false} & \text{if } a = 1 \\
  \text{even}(a-2) & \text{otherwise}
  \end{cases}
\end{align*}
\]

Russian peasants’ algorithm
\[
\begin{align*}
\text{Product} (a, b) &\quad \text{result} \leftarrow 0 \\
\text{while } a > 0 &\quad \text{if } a \text{ is odd} \\
&\quad \text{add } b \text{ to result} \\
&\quad a \leftarrow \lfloor a \div 2 \rfloor \\
&\quad b \leftarrow b \times 2 \\
\text{Return result}
\end{align*}
\]

Example: \( 13 \times 5 = 65 \)
\[
\begin{align*}
1 &\rightarrow 0 \\
13 &\rightarrow 5 \\
6 &\rightarrow 10 \\
3 &\rightarrow 20 \\
1 &\rightarrow 40 \\
0 &\rightarrow 80
\end{align*}
\]

Non-algorithmic computations
- The process diagrammed at right will never terminate if user continues to input a non-zero value

Object-oriented design
- Object-oriented design focuses on classes of objects and their attributes and behaviors
2. Design

Unified Modeling Language
• A new standard graphical notation for system specification and design
• Motivated by need to depict interactions between systems and their environments, initiated by external actors
• Diagrams include use-case, activity, class, state, interaction
• Supports an object-oriented methodology

Software projects start with problem descriptions
• What service must the software provide? 
  Example: Process customer orders from catalog
• What assumptions are made? 
  Example: Some customers may find Web access convenient
• What risks are involved? 
  Example: Some users are inexperienced with Web access

Use cases and system specification
• Use case: A typical interaction between system and an actor in its environment
• Actor: A role, such as customer, manager, supplier, salesperson
• Actors initiate use cases
• Example of two uses cases (UML use-case diagram):

UML activity diagrams
• Depict order in which steps occur
• Examples:
  - A decision point (branching)
  - A fork (2 forms A join (execution display at once) converges)

Four CS I skills objectives
• Vague specification → precise specification (analysis)
• Problem specification → flowchart (design)
• Flowchart → trace (design verification)
• Flowchart → program (coding)
• Program → trace (testing)

Discussion problems
(a) Tell what control structures are required.
(b) Solve, using pseudocode or flowchart.
1. Calculate the cost of transporting a crate of goods, given user input for the dimensions of the crate, the cost per mile per cubic foot, and the number of miles.
2. Input and echo retail-inventory part numbers and quantities until the user inputs part # 0.
3. Input the dimensions of a rectangular plot of land and tell whether or not its area is over an acre (~42,000 square feet).
Loop discussion problems

1. Design and trace an algorithm that will loop to accept input of exactly three pairs of integers \((a, b)\) and compute and display the value of \(a - b\) for each input pair.

2. Design and trace a loop to compute and display the product of two input non-negative integers. Display nothing if input includes a negative number.