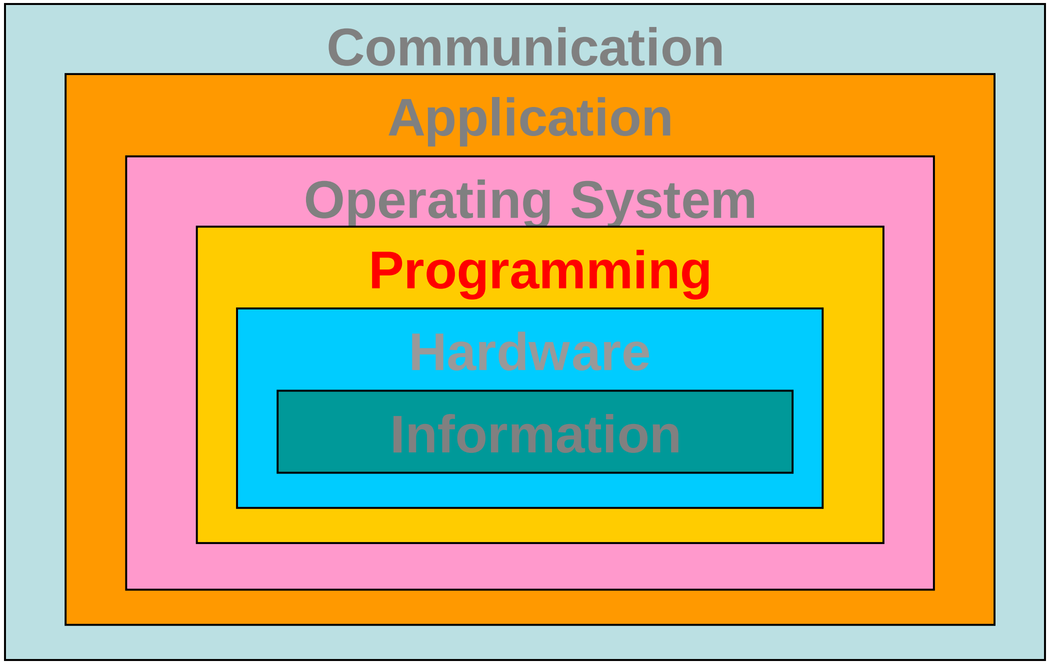
Introduction to Computing

Section 4 – Programming Layer



Problem Solving

- G. Polya wrote *How to Solve It: A New Aspect of Mathematical Method*
- His “How to Solve It” list is quite general
  - Written in the context of solving mathematical problems
  - The list becomes applicable to all types of problems

Ask Questions...

- …to understand the problem
  - *What do I know about the problem?*
  - *What is the information that I have to process in order the find the solution?*
  - *What does the solution look like?*
  - *What sort of special cases exist?*
  - *How will I recognize that I have found the solution?*

Look for Familiar Things

SCSE

- You should never reinvent the wheel
- In computing, you see certain problems again and again in different guises
- A good programmer sees a task, or perhaps part of a task (a subtask), that has been solved before and plugs in the solution

Divide and Conquer

- Break up a large problem into smaller units that we can handle
  - Applies the concept of abstraction
  - The divide-and-conquer approach can be applied over and over again until each subtask is manageable

Algorithms

SCSE

- **Algorithm** A set of instructions for solving a problem or sub-problem in a finite amount of time using a finite amount of data
- The instructions must be unambiguous

Computer Problem-Solving

Algorithm Development Phase		
	Analyze	Understand (define) the problem.
	Propose algorithm	Develop a logical sequence of steps to be used to solve the problem.
	Test algorithm	Follow the steps as outlined to see if the solution truly solves the problem.
Implementation Phase		
	Code	Translate the algorithm (the general solution) into a programming language.
	Test	Have the computer follow the instructions. Check the results and make corrections until the answers are correct.
Maintenance Phase		
	Use	Use the program.
	Maintain	Modify the program to meet chaining requirements or to correct any errors.

Figure 6.2 **The computer problem-solving process**

Methodology
for designing algorithms

- Analyze the problem
- List the main Tasks
- Write the remaining Modules
- Re-sequence and revise as necessary

Top-Down Design

SCSE

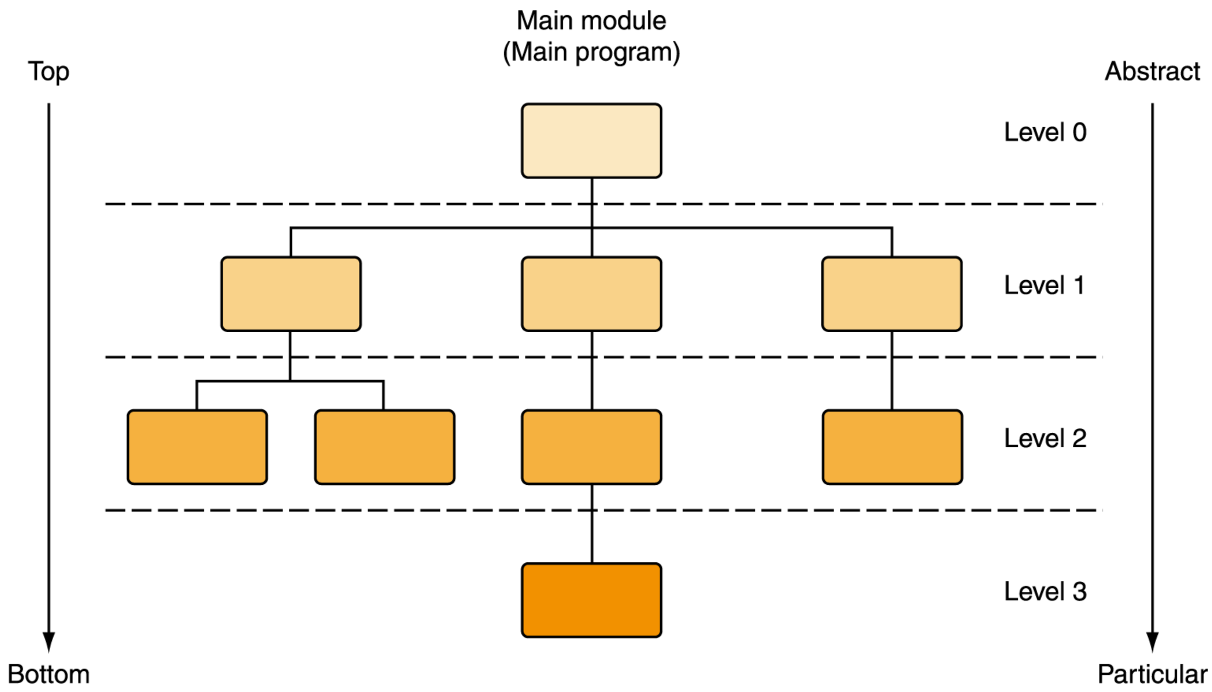


Figure 6.5
An example of top-down design

- This process continues for as many levels as it takes to expand every task to the smallest details
- A step that needs to be expanded is an abstract step

Pseudocode

SCSE

- Uses a mixture of English and formatting to make the steps in the solution explicit

```
While (the quotient is not zero)
        Divide the decimal number by the new base
        Make the remainder the next digit to the left in the answer
        Replace the original decimal number with the quotient
```

Algorithms

SCSE

An algorithm:

- an ordered sequence of precisely defined instructions that performs some task in a finite amount of time.

- must have ability to alter the order of its instructions using a *control structure*.

Algorithm operations: sequential operations, conditional operations, iterative operations (loops)

Algorithms

SCSE

Sequential operations: executed in order.

Conditional operations: first ask a question to be answered with a true/false answer and the select the next instruction based on the answer.

Iterative operations (loops): repeat the execution of a block of instruction

Algorithms

SCSE

Sequential Operations

Compute the perimeter p and the area A of a triangle whose sides are a, b, c. The formulas are:

$$p = a + b + c \qquad s = \frac{p}{2} \qquad A = \sqrt{s(s-a)(s-b)(s-c)}$$

1. Enter the side lengths a, b, and c.
2. Compute the perimeter p: p=a+b+c
3. Compute the semi perimeter s: s = p/2
4. Compute the area A.
5. Display the results p and A.
6. Stop

Conditional Operations

Given the (x,y) coordinates of a point, compute its polar coordinates (r,θ), where

$$r = \sqrt{x^2 + y^2} \qquad \theta = \tan^{-1}\left(\frac{y}{x}\right)$$

1. Enter the coordinates x and y.
2. Compute the hypoteneuse r. $r = \sqrt{x^2 + y^2}$
3. Compute the angle θ

3.1. If x≥0: $\theta = \tan^{-1}\left(\frac{y}{x}\right)$

3.2. Else: $\theta = \tan^{-1}\left(\frac{y}{x}\right) + pi$

4. Convert the angle to degrees. $\theta = \theta * 180 / pi$
5. Display the results r and θ.
6. Stop

Iterative Operations

Determine how many terms are required for the sum of the series $10k^2 - 4k + 2$, k=1, 2, 3, … to exceed 20,000. What is the sum for this many terms.

Because we do not know how many times we must evaluate the expression $10k^2 - 4k + 2$, we use a "while" loop.

1. Initialize the total to zero.
2. Initialize the counter to zero.
3. While the total is less than 20,000 compute the total.

3.1. Increment the counter by 1: k=k+1;

3.2. Update the total: $total = 10 * k^2 - 4 * k + 2 + total$

4. Display the current value of the counter.

5. Display the value of the total.

6. Stop.

Algorithms with Simple Variables

- An algorithm with selection

Ex: What dress is appropriate for a given outside temperature with four options:

- Shorts if it is hot
- Short sleeves if it is nice but not too hot
- A light jacket if the temperature is chilly
- Heavy coat if it is cold
- If the temperature is below freezing, stay inside

Algorithms with Simple Variables

SCSE

The top-level (main) module:

1. Write “Enter the temperature”
2. Read the temperature

Not need further decomposing

3. Determine dress:

- List all cases and define the corresponding temperatures

  hot: >90, nice: >70, chilly: >50, cold: >32

- Write the pseudocode for “Determine dress”

Algorithms with Simple Variables

SCSE

Determine dress (pseudocode)

```
IF (temperature >90)
        Write "so hot: wear shorts"
ELSE IF (temperature >70)
        Write "Ideal temperature: short sleeves are fine"
ELSE IF (temperature >50)
        Write "A little chilly: wear a light jacket"
ELSE IF (temperature >32)
        Write "so cold: wear a heavy coat"
ELSE
        Write "Stay inside"
```

Algorithms with Simple Variables

SCSE

- An algorithm with repetition (count controlled and event controlled)

➢ Count controlled loops: repeats a process a specified number of times.

➢ Event controlled loops: the number of repetition is controlled by an event that occurs within the body of the loop itself.

Algorithms with Simple Variables

SCSE

Count controlled loops:

Three distinct parts:

1. Initialization: loop control variable
2. Testing: loop control variable reaches a predetermined value?
3. Incrementation: loop variable is incremented by a value?

Algorithms with Simple Variables

SCSE

Read limit //*Input data*

Set count to 0 //*Initialize count to 0*

WHILE (count < limit) //*Test*

    …….. //*Body of the loop*

    Set count to count + 1 //*Increment*

……… //*Statement(s) following loop*

Algorithms with Simple Variables

SCSE

Example: A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz

Set total to zero
Set grade counter to one

While grade counter is less than or equal to ten
Input the next grade
Add the grade into the total
Add one to the grade counter

Set the class average to the total divided by ten
Print the class average

Algorithms with Simple Variables

Event controlled loops:

Three distinct parts:

1. The event must be initialized
2. The event must be tested
3. The event must be updated

Algorithms with Simple Variables

SCSE

Read and sum data values until a negative value is read:

1. What is the event?

→ Reading a positive value.

2. How do we initialize the event?

→Reading the first data value, testing the value to determine whether its is positive and enter the loop if it is.

3. How do we update the event?

→Reading the next data value.

Algorithms with Simple Variables

Example:

Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.

- Unknown number of students
- How will the program know to end?

$\Rightarrow$ Use sentinel value

- Also called signal value, dummy value, or flag value
- Indicates “end of data entry.”
- Loop ends when user inputs the sentinel value
- Sentinel value chosen so it cannot be confused with a regular input (such as **`-1`** in this case)

Algorithms with Simple Variables

SCSE

Nested structures:

A structure in which one control structure is embedded within another

Algorithms with Simple Variables

SCSE

- Problem
  - A college has a list of test results (**1** = pass, **2** = fail) for 10 students
  - Write a program that analyzes the results
    - If more than 8 students pass, print "Raise Tuition"
- Notice that
  - The program must process 10 test results
    - Counter-controlled loop will be used
  - Two counters can be used
    - One for number of passes, one for number of fails
  - Each test result is a number—either a **1** or a **2**
    - If the number is not a **1**, we assume that it is a **2**

Algorithms with Simple Variables

SCSE

- Top level outline

  Analyze exam results and decide if tuition should be raised

- First Refinement

  Initialize variables

  Input the ten quiz grades and count passes and failures

  Print a summary of the exam results and decide if tuition should be raised

- Refine *Initialize variables* to

  Initialize passes to zero

  Initialize failures to zero

  Initialize student counter to one

Algorithms with Simple Variables

SCSE

- Refine *Input the ten quiz grades and count passes and failures* to

  While student counter is less than or equal to ten
  Input the next exam result
  If the student passed
  Add one to passes
  else
  Add one to failures
  Add one to student counter

- Refine *Print a summary of the exam results and decide if tuition should be raised* to

  Print the number of passes
  Print the number of failures
  If more than eight students passed
  Print “Raise tuition”

Algorithms with Simple Variables

PRACTICE

1. Write an algorithm to input a number and find its square root if it is positive and power 2 in case of negative.
2. Write an algorithm to input a number and find its power 2 if it belongs to [0,5], square root if it is greater than 5 and no change in case of negative.
3. Write an algorithm to solve the first order equation.
4. Write an algorithm to calculate sum of all even numbers from 1 to n, n is optional.

C Program: Basic concepts

SCSE

Comments

- Text surrounded by `/*` and `*/` is ignored by computer
- Used to describe program

`#include <stdio.h>`

- Preprocessor directive
  - Tells computer to load contents of a certain file
- `<stdio.h>` allows standard input/output operations

C Program: Basic concepts

SCSE

int main()

C++ programs contain one or more functions, exactly one of which must be **main**

Parenthesis used to indicate a function

int means that **main** "returns" an integer value

Braces (**{** and **}**) indicate a block: the bodies of all functions must be contained in braces

C Program: Basic concepts

int integer1, integer2, sum;

- Declaration of variables
  - Variables: locations in memory where a value can be stored
- **int** means the variables can hold integers (**-1**, **3**, **0**, **47**)
- Variable names (identifiers)
  - **integer1**, **integer2**, **sum**
  - Identifiers: consist of letters, digits (cannot begin with a digit) and underscores(_)
    - Case sensitive
- Declarations appear before executable statements
  - If an executable statement references and undeclared variable it will produce a syntax (compiler) error

C Program: Basic concepts

printf(“Enter the first integer : \n");

Instructs computer to perform an action

Specifically, prints the string of characters within quotes (**“ ”**)

Entire line called a statement

All statements must end with a semicolon (**;**)

Escape character (****)

Indicates that printf should do something out of the ordinary

\n is the newline character

C Program: Basic concepts

SCSE

scanf("%d", &integer1);

- Obtains a value from the user

  scanf uses standard input (usually keyboard)

- This **scanf** statement has two arguments
  - ✓**%d** - indicates data should be a decimal integer
  - ✓**&integer1** - location in memory to store variable
  - ✓**&** is confusing in beginning – for now, just remember to include it with the variable name in **scanf** statements
- When executing the program the user responds to the **scanf** statement by typing in a number, then pressing the *enter* (return) key

A Simple C Program: Printing a Line of Text

```
/* Fig. 2.1: fig02_01.c
   A first program in C */
#include <stdio.h>

int main()
{
   printf( "Welcome to C!\n" );

   return 0;
}
```

A Simple C Program: Addition program

```c
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

int main()
{
          int integer1, integer2, sum;       /* declaration */

          printf( "Enter first integer\n" );  /* prompt */
          scanf( "%d", &integer1 );           /* read an integer */
          printf( "Enter second integer\n" ); /* prompt */
          scanf( "%d", &integer2 );           /* read an integer */
          sum = integer1 + integer2;          /* assignment of sum */
          printf( "Sum is %d\n", sum );       /* print sum */

 return 0;  /* indicate that program ended successfully */
}
```

Programming

SCSE

PRACTICE

1. Develop a C program to input a number and find its square root if it is positive and power 2 in case of negative.
2. Develop a C program to input a number and find its power 2 if it belongs to [0,5], square root if it is greater than 5 and no change in case of negative.
3. Develop a C program to solve the first order equation.
4. Develop a C program to calculate sum of all even numbers from 1 to n, n is optional.

Algorithms with Composite Variables

- ARRAYS

A collection of homogeneous items in which individual items are accessed by their place within the collection (index)

Most programming languages start at index 0.

EX: if the array is called *numbers,* we access each value by *numbers[position]*

Position is also the index.

Algorithms with Composite Variables

SCSE

The algorithm to put values into the places in an array

```
integer numbers[10]
//Declare numbers to hold 10 integer values
Write “Enter 10 integer numbers, one per line”
Set position to 0 //Set variable position to 0
WHILE (position <10)
        Read in numbers[position]
        Set position to position + 1
//Continue with processing
```

Algorithms with Composite Variables

SCSE

Algorithms with arays:

1. Searching
2. Sorting
3. Processing

Sequential search:

Read in array of values
Write “Enter value for which to search”
Read searchItem
Set found to TRUE if searchItem is there
IF (found)
    Write “Item is found”
ELSE
    Write “Item is not found”

[0]	60
[1]	75
[2]	95
[3]	80
[4]	65
[5]	90
…	…
[l-1]	..

Unordered array

[0]	60
[1]	65
[2]	75
[3]	80
[4]	90
[5]	95
…	…
[l-1]	…

Sorted array

Read in array of values:
Write “How many values?”
Read length
Set index to 0
WHILE (index < length)
    Read data[index]
    Set index to index+1

Set found to TRUE if searchItem is there
Set index to 0
Set found to FALSE
WHILE (index<length AND NOT found)
    IF (data[index] equals searchItem)
        Set found to TRUE
    ELSE IF (data[index]>searchItem)
        Set index to length
    ELSE
        Set index to index+1

Binary search:

Looking for an item in an already sorted list by eliminating large portions of the data on each comparison.

Boolean Binary Search

```
Set first to 0
Set last to length-1
Set found to FALSE
WHILE (first<=last AND NOT found)
        Set middle to (first+last)/2
        IF (item equals data[middle])
                Set found to TRUE
        ELSE
                IF (item<data[middle])
                        Set last to middle - 1
                ELSE
                        Set first to middle+1
Return found
```

[0]	ant
[1]	cat
[2]	chicken
[3]	cow
[4]	deer
[5]	dog
[6]	fish
[7]	goat
[8]	horse
[9]	rat
[10]	snake
….	

Sorted list, length=11

Searching for cat

First	Last	Middle	Comparison
0	10	5	cat<dog
0	4	2	cat<chicken
0	1	0	cat>ant
1	1	1	cat=cat **Return: TRUE**

Searching for fish

First	Last	Middle	Comparison
0	10	5	fish>dog
6	10	8	fish<horse
6	7	6	fish=fish **Return: TRUE**

Searching for zebra

First	Last	Middle	Comparison
0	10	5	zebra>dog
6	10	8	zebra>horse
9	10	9	zebra>rat
10	10	10	zebra>snake
11	10		**first>last** **Return: FALSE**

[0]	ant
[1]	cat
[2]	chicken
[3]	cow
[4]	deer
[5]	dog
[6]	fish
[7]	goat
[8]	horse
[9]	rat
[10]	snake
….	

Sorted list, length=11

Selection sort:

[0]	Sue
[1]	Cora
[2]	Beth
[3]	Ann
[4]	June

[0]	Ann
[1]	Cora
[2]	Beth
[3]	Sue
[4]	June

[0]	Ann
[1]	Beth
[2]	Cora
[3]	Sue
[4]	June

[0]	Ann
[1]	Beth
[2]	Cora
[3]	Sue
[4]	June

[0]	Ann
[1]	Beth
[2]	Cora
[3]	June
[4]	Sure

Selection sort

Set firstUnsorted to 0

WHILE (firstUnsorted < length-1)

    Find smallest unsorted item

    Swap firstUnsorted item with the smallest

    Set firstUnsorted to firstUnsorted+1

Selection sort (cont):

SCSE

Find smallest unsorted item

```
Set indexOfSmallest to firstUnsorted
Set index to firstUnsorted+1
WHILE (index<=length-1)
        IF (data[index]<data[indexOfSmallest])
                Set indexOfSmallest to index
        Set index to index+1
```

Swap firstUnsorted with the smallest

```
Set tempItem to data[firstUnsorted]
Set data[firstUnsorted] to data[indexOfSmallest]
Set data[indexOfSmallest] to tempItem
```

Bubble sort:

Starting with the last array element, compare successive pairs of elements, swapping them whenever the bottom element of pair is smaller than the one above it.

First iteration

[0]	Phil
[1]	Al
[2]	John
[3]	Jim
[4]	Bob

[0]	Phil
[1]	Al
[2]	John
[3]	Bob
[4]	Jim

[0]	Phil
[1]	Al
[2]	Bob
[3]	John
[4]	Jim

[0]	Phil
[1]	Al
[2]	Bob
[3]	John
[4]	Jim

[0]	Al
[1]	Phil
[2]	Bob
[3]	John
[4]	Jim

Remaining iteration

[0]	Al
[1]	Phil
[2]	Bob
[3]	John
[4]	Jim

[0]	Al
[1]	Bob
[2]	Phil
[3]	Jim
[4]	John

[0]	Al
[1]	Bob
[2]	Jim
[3]	Phil
[4]	John

[0]	Al
[1]	Bob
[2]	Jim
[3]	John
[4]	Phil

Bubble sort (cont):

SCSE

Bubble sort

```
Set firstUnsorted to 0
Set swap to TRUE
WHILE (firstUnsorted <length-1 AND swap)
        Set swap to FALSE
        "Bubble up" the smallest item in unsorted part
        Set firstUnsorted to firstUnsorted+1
```

Bubble up

```
Set index to lenghth-1
WHILE (index>firstUnsorted+1)
        IF (data[index]<data[index-1])
                Swap data[index] and data[index-1]
                Set swap to TRUE
        Set index to index-1
```

Insertion sort:

[0]	Phil
[1]	John
[2]	Al
[3]	Jim
[4]	Bob

[0]	John
[1]	Phil
[2]	Al
[3]	Jim
[4]	Bob

[0]	Al
[1]	John
[2]	Phil
[3]	Jim
[4]	Bob

[0]	Al
[1]	Jim
[2]	John
[3]	Phil
[4]	Bob

[0]	Al
[1]	Bob
[2]	Jim
[3]	John
[4]	Phil

Insertion sort

```
Set current to 1  // current is the item being inserted into the sorted portion
WHILE (current < length)
        Set index to current
        Set placeFound to FALSE
        WHILE (index>0 AND NOT placeFound)
                IF (data[index] < data[index-1])
                        Swap data[index] and data[index-1]
                        Set index to index-1
                ELSE
                        Set placeFound to TRUE
        Set current to current+1
```

Recursive algorithms

Recursion:

The ability of an algorithm to call itself

- Recursive factorial
- Recursive binary search

Recursive factorial

<u>Factiorial of N</u>:

N!=1.2.3.4.5.6…..N=N*(N-1)!

Factorial of 0 is 1.

Recursive factorial	Factorial(N)
Write “Enter N”	IF (N equals to 0)
Read N	RETURN
Set result to Factorial(N)	ELSE
Write result + “is the factorial of” +N	RETURN N*Factorial(N)

Recursive Binary Search

SCSE

```
BinarySearch (first, last)
IF (first>last)
        RETURN FALSE
ELSE
        Set middle to (first+last)/2
        IF (item equals data[middle])
                RETURN TRUE
        ELSE
                IF (item<data[middle]
                        BinarySearch (first, middle-1)
                ELSE
                        BinarySearch(middle+1, last)
```

Algorithms with Composite Variables

- RECORDS

A named heterogeneous groups of items in which individual items are accessed by name.

“Heterogeneous”: elements in the collection do not have to be the same.

Collections: integers, real values, strings, other types of data.

RECORDS

	Employee
Name	
Age	
hourlyWage	

Store values into the fields of the record

Employee employee   //Declare an Employee variable

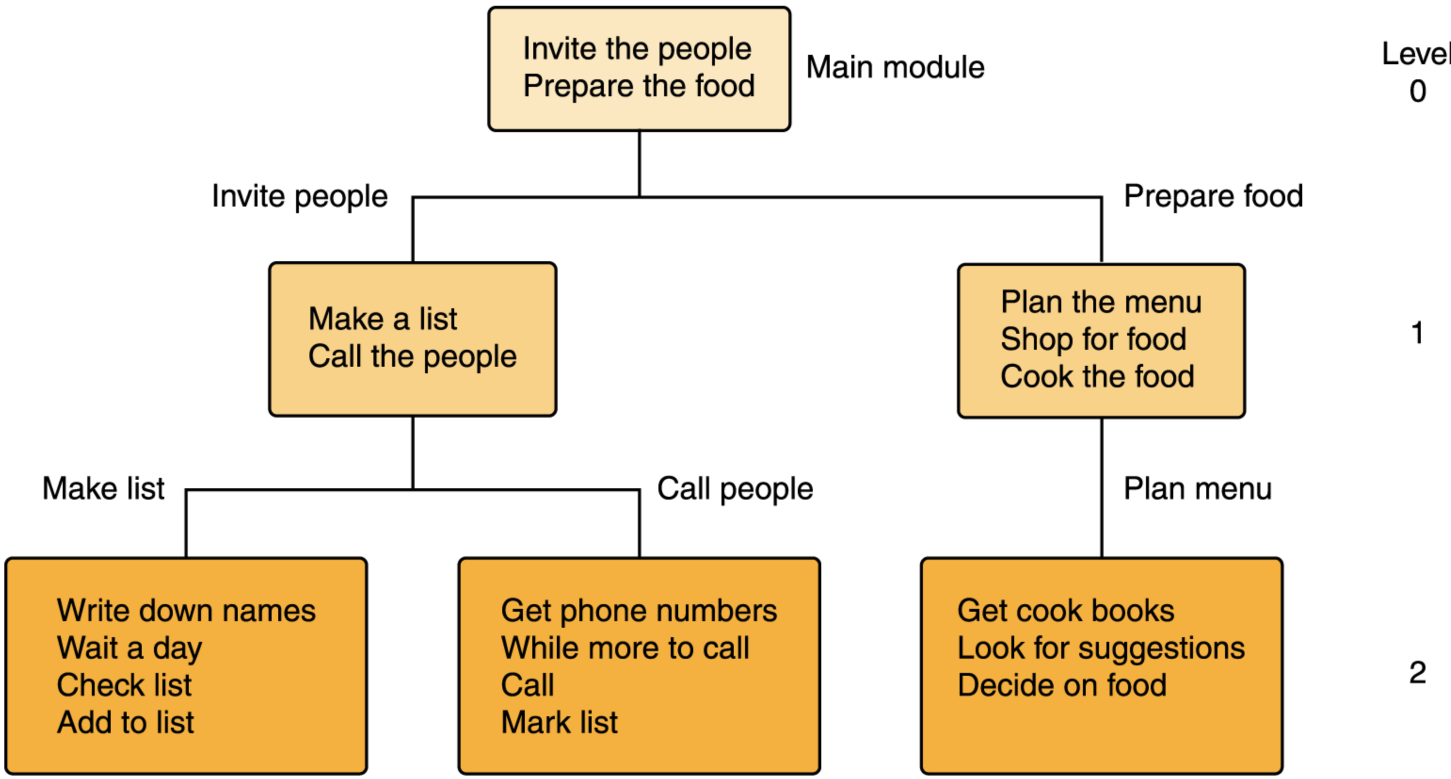
Set employee.name to “Nguyen Van A”

Set employee.age to 32

Set employee.hourlyWage to 27.50

A General Example

- Planning a large party



A Computer Example

SCSE

- Problem
  - Create an address list that includes each person's name, address, telephone number, and e-mail address
  - This list should then be printed in alphabetical order
  - The names to be included in the list are on scraps of paper and business cards

A Computer Example

SCSE

Main — Level 0

Enter names into list
Fill in missing data
Put list into alphabetical order
Print the list

Enter names into list — Level 1

Prompt for and enter names — includes other data as well
Insert names into list

A Computer Example

SCSE

Prompt for and enter names **Level 2**

Write "To any of the prompts below, if the information is not known, just press return."
While (more names)
    Write "Enter the last name, a comma, a blank, and the first name; press return."
    Read lastFirst
    Write "Enter street number and name; press return."
    Read street
    Write "Enter city, a comma, a blank, and state; press return."
    Read cityState
    Write "Enter area code and 7-digit number; press return."
    Read telephone
    Write "Enter e-mail; press return."
    Read eMail

A Computer Example

SCSE

Fill in missing data **Level 1**

```
Write "To any of the prompts below, if the information is still not known,
      just press return."
Get a name from the list
While there are more names
   Get a lastFirst
   Write lastFirst
   If (street is missing)
      Write "Enter street number and name; press return."
      Read street
   If (telephone is missing)
      Write "Enter area code and 7-digit number; press return."
      Read telephone
   If (eMail is missing)
      Write "Enter e-mail; press return."
   Get a name from the list
```

A Computer Example

SCSE

Put list in alphabetical order Level 3

Sort list on lastFirst field

Print the list

Write "The list of names, addresses, telephone numbers, and e-mail addresses follows:"

Get a name from the list

While (there are more names)

    Write lastFirst

    Write street

    Write cityState

    Write e-Mail

    Write a blank line

    Get a name from the list

Testing the Algorithm

SCSE

- The process itself must be tested
- Testing at the algorithm development phase involves looking at each level of the top-down design

Testing the Algorithm

- **Desk checking** Working through a design at a desk with a pencil and paper
- **Walk-through** Manual simulation of the design by the team members, taking sample data values and simulating the design using the sample data
- **Inspection** One person (not the designer) reads the design (handed out in advance) line by line while the others point out errors

Object-Oriented Design

SCSE

- A problem-solving methodology that produces a solution to a problem in terms of self-contained entities called *objects*

- **Object** A thing or entity that makes sense within the context of the problem

  For example, a student

Object-Oriented Design

SCSE

- A group of similar objects is described by an **object class**, or **class**
- A class contains fields that represent the properties and behaviors of the class
  - A **field** can contain data value(s) and/or methods (subprograms)
  - A **method** is a named algorithm that manipulates the data values in the object

Relationships Between Classes

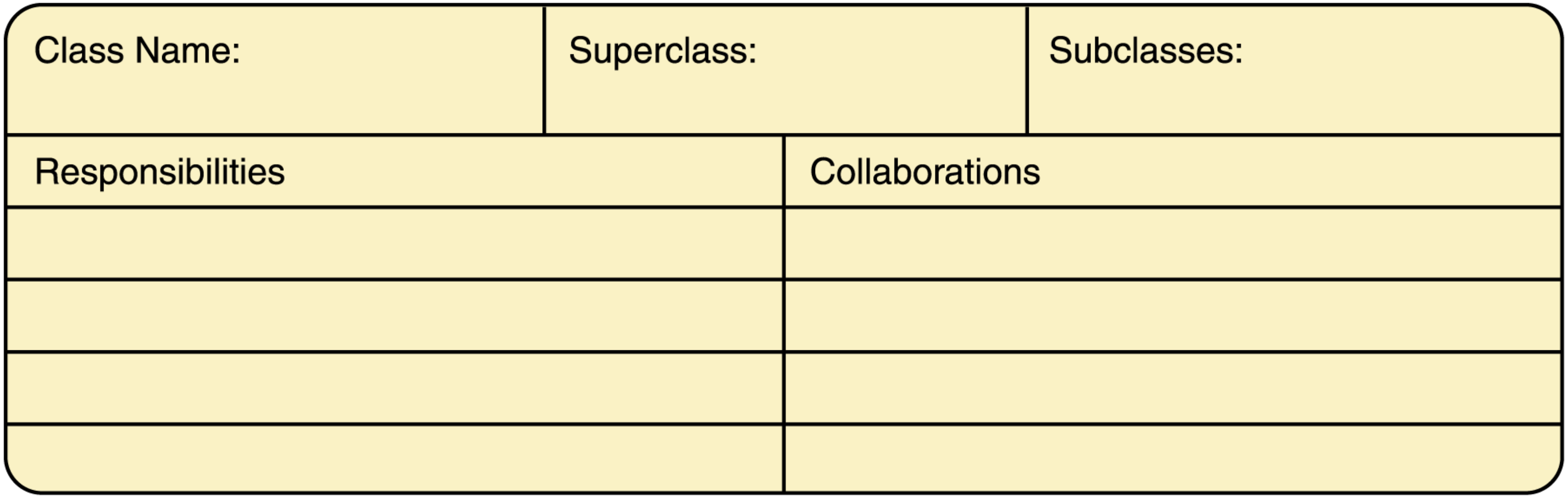
- Containment
  - “part-of”
  - An address class may be part of the definition of a student class
- Inheritance
  - Classes can inherit data and behavior from other classes
  - “is-a”

Object-Oriented Design Methodology

SCSE

- Four stages to the decomposition process
  - **Brainstorming**
  - **Filtering**
  - **Scenarios**
  - **Responsibility algorithms**

CRC Cards


Class Name:
Superclass:
Subclasses:
Responsibilities
Collaborations

Brainstorming

SCSE

- A group problem-solving technique that involves the spontaneous contribution of ideas from all members of the group
  - All ideas are potential good ideas
  - Think fast and furiously first, and ponder later
  - A little humor can be a powerful force
- Brainstorming is designed to produce a list of candidate classes

Filtering

SCSE

- Determine which are the core classes in the problem solution
- There may be two classes in the list that have many common attributes and behaviors
- There may be classes that really don't belong in the problem solution

Scenarios

- Assign responsibilities to each class
- There are two types of responsibilities
  - What a class must know about itself (knowledge responsibilities)
  - What a class must be able to do (behavior responsibilities)

Scenarios

- Each class **encapsulates** its data but shares their values through knowledge responsibilities.

- **Encapsulation** is the bundling of data and actions in such a way that the logical properties of the data and actions are *separated from the implementation details*

Responsibility Algorithms

SCSE

- The algorithms must be written for the responsibilities
  - Knowledge responsibilities usually just return the contents of one of an object's variables
  - Action responsibilities are a little more complicated, often involving calculations

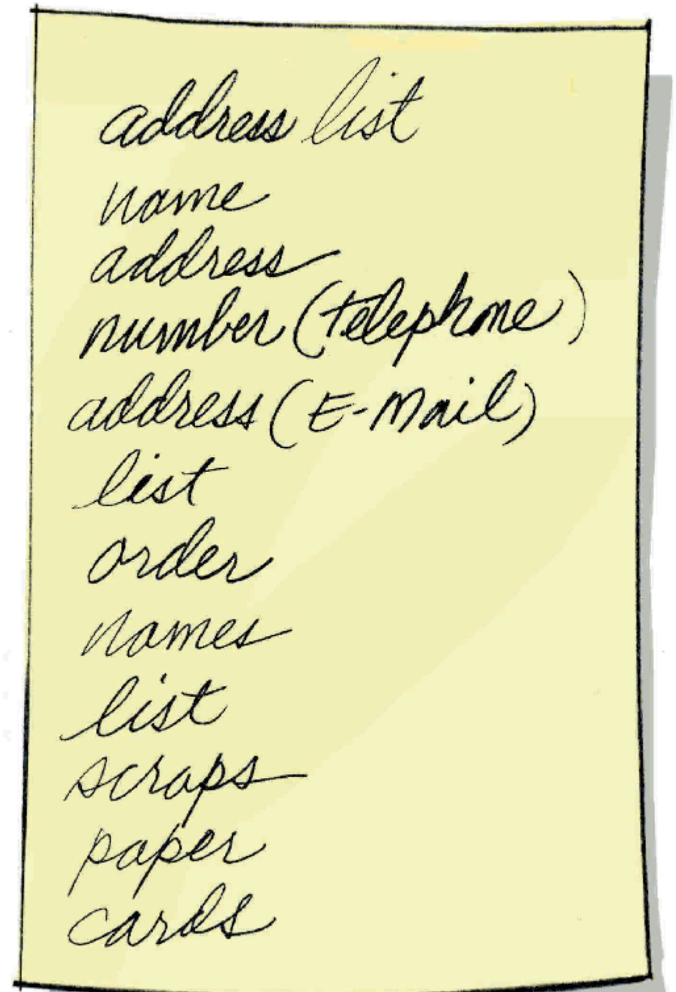
Computer Example

- Let’s repeat the problem-solving process for creating an address list
- Brainstorming and filtering
  - Circling the nouns and underlining the verbs

Create an address list that includes each person's name, address, telephone number, and e-mail address. This list should then be printed in alphabetical order. The names to be included in the list are on scraps of paper and business cards.

Computer Example

- First pass at a list of classes



Computer Example

SCSE

- Filtered list

CRC Cards

Class Name: Person	Superclass:	Subclasses:

Responsibilities	Collaborations
Initialize itself (name, address, telephone, e-mail)	Name, Address, Telephone, E-mail
Print	Name, Address, Telephone, E-mail

Class Name: Name	Superclass:	Subclasses:

Responsibilities	Collaborations
Initialize itself (name)	String
Print itself	String

Responsibility Algorithms

SCSE

Initialize

name.Initialize()
address.Initialize()
telephone.Initialize()
email.Initialize()

Print

name.Print()
address.Print()
telephone.Print()
email.Print()

Information Hiding/Abstraction

SCSE

- **Information Hiding** and **Abstraction** are two sides of the same coin.
  - **Information Hiding** The practice of hiding the details of a module with the goal of controlling access to the details of the module.
  - **Abstraction** A model of a complex system that includes only the details essential to the viewer.

Information Hiding/Abstraction

SCSE

- Abstraction is the result with the details hidden
  - **Data abstraction** Separation of the logical view of data from their implementation.
  - **Procedural abstraction** Separation of the logical view of actions from their implementation.
  - **Control abstraction** Separation of the logical view of a control structure from its implementation.

Programming Languages

SCSE

- Instructions written in a **programming language** can be *translated* into the instructions that a computer can execute directly
- **Program** A meaningful sequence of instructions for a computer
  - **Syntax** The part that says how the instructions of the language can be put together
  - **Semantics** The part that says what the instructions mean

Review

- Describe the computer problem-solving process.
- Distinguish between a simple type and a composite type
- Simple C programs
- Describe three composite data-structuring mechanisms
- Recognize a recursive problem and write a recursive algorithm to solve it
- Distinguish between an unsorted array and a sorted array

Review

- Distinguish between a selection & an insertion sort
- Describe Quicksort algorithm
- Apply the selection sort, the bubble sort, insertion sort, and Quicksort to an array of items by hand
- Apply the binary search algorithm
- Demonstrate your understanding of the algorithms in this chapter by hand-simulating them with a sequence of items