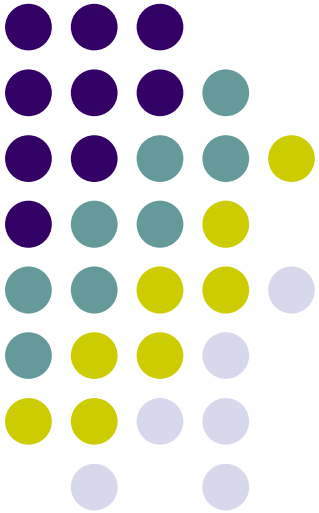
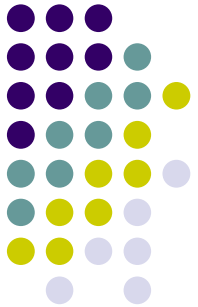
CSI2441: Applications Development

Lecture 2

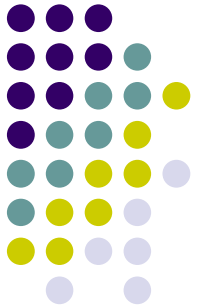
Understanding Program Structure





Objectives

- Describe the features of unstructured spaghetti code
- Describe the three basic structures – sequence, selection, and loop
- Use a priming read
- Appreciate the need for structure
- Recognize structure
- Describe three special structures – case, do-while, and do-until



Understanding Unstructured Spaghetti Code

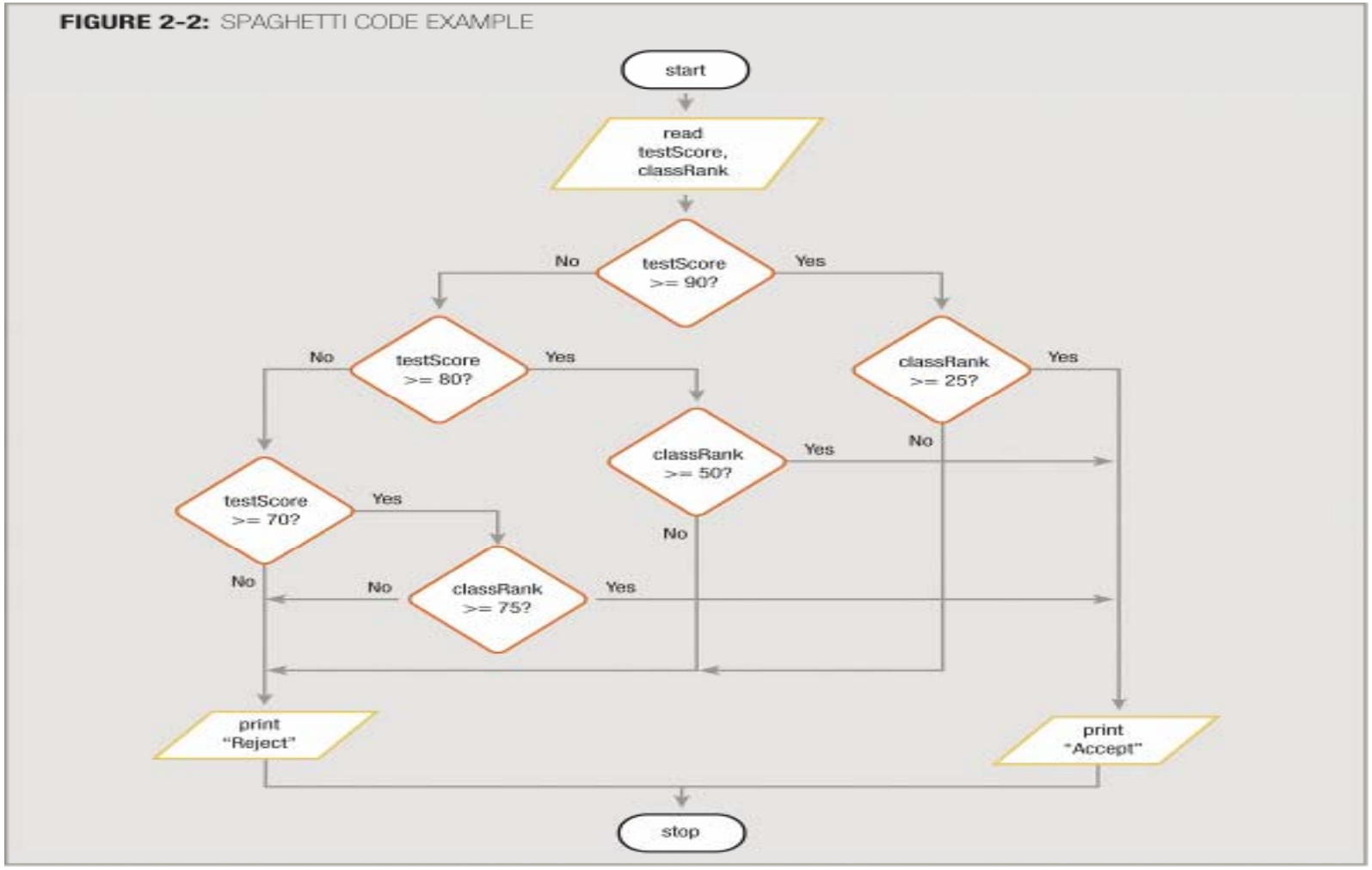
- **Spaghetti code** – logically snarled program statements
- Can be the result of poor program design
  Example: college admissions criteria

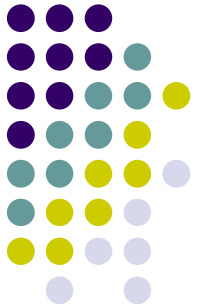
TABLE 2-1: ADMISSION REQUIREMENTS

Test score	High-school rank
90–100	Upper 75 percent (from 25th to 100th percentile)
80–89	Upper half (from 50th to 100th percentile)
70–79	Upper 25 percent (from 75th to 100th percentile)

Understanding Unstructured Spaghetti Code (continued)

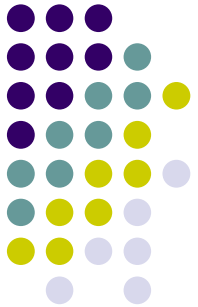
FIGURE 2-2: SPAGHETTI CODE EXAMPLE





Understanding Unstructured spaghetti Code (continued)

- Spaghetti code programs often work, but are difficult to read and maintain
- Is usually quicker to write – ie ‘code as you go’
- Convoluted logic usually requires more code
- Developers can often think ‘just testing the concept – will fix it later’ – which rarely if ever happens
- Nightmare to fix and unravel later
- Correctly analysing and structuring a program before you start coding takes a little extra time
- Correctly re-structuring a program after it has been coded can take far far longer and may never be fully successful

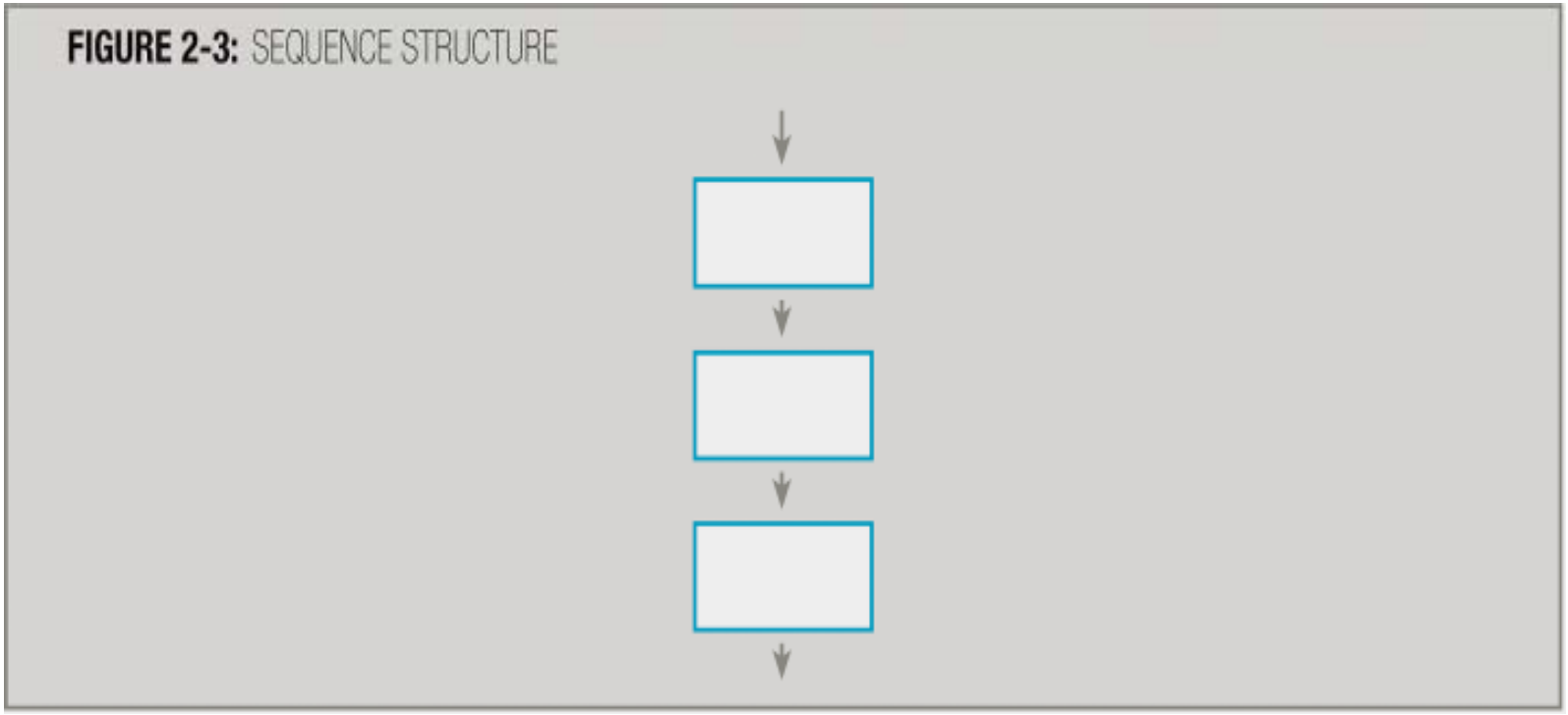


Understanding the Three Basic Structures

- **Structure**: a basic unit of programming logic
- Any program can be constructed from only three basic types of structures
  - Sequence
  - Selection
  - Loop

Understanding the Three Basic Structures (continued)

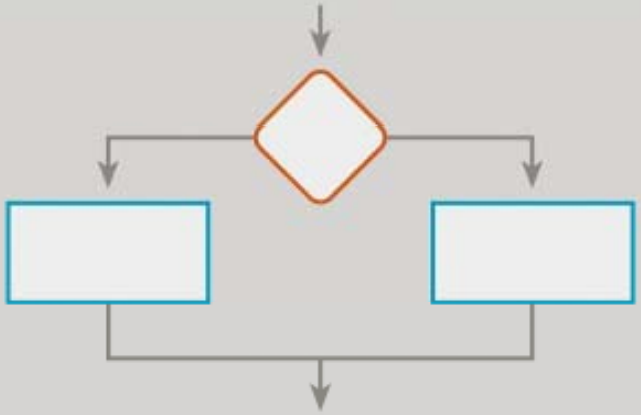
- **Sequence structure**
  - A set of instructions, performed sequentially with no branching

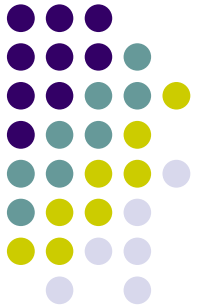


Understanding the Three Basic Structures (continued)

- **Selection structure**
  - Asks a question, then takes one of two possible courses of action based on the answer
  - Also called a **decision structure** or an **if-then-else**

FIGURE 2-4: SELECTION STRUCTURE





Understanding the Three Basic Structures (continued)

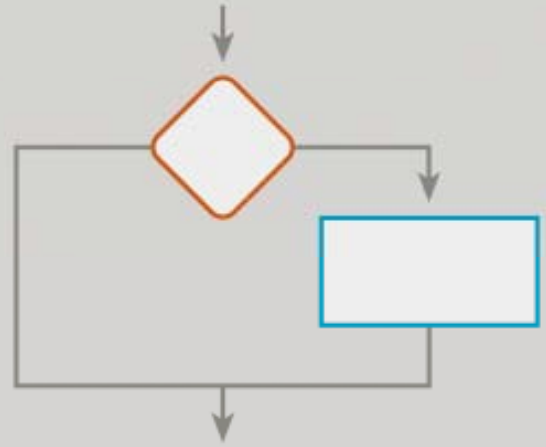
- **Dual-alternative if**: contains two alternatives

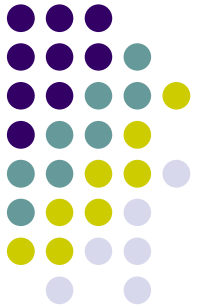
```
if hoursWorked is more than 40 then
   calculate regularPay and overtimePay
else
   calculate regularPay
```

Understanding the Three Basic Structures (continued)

- **Single-alternative if**: contains one alternative
- Or, a one-sided condition

FIGURE 2-5: SINGLE-ALTERNATIVE DECISION STRUCTURE





Understanding the Three Basic Structures (continued)

- **Single-alternative if**

```
if employee belongs to dentalPlan then
    deduct $40 from employeeGrossPay
```

- Else clause is not required
- **Null case**: situation where nothing is done
- Some novice developers sometimes write long series of single-alternate if statements to avoid complex if-then-else structures

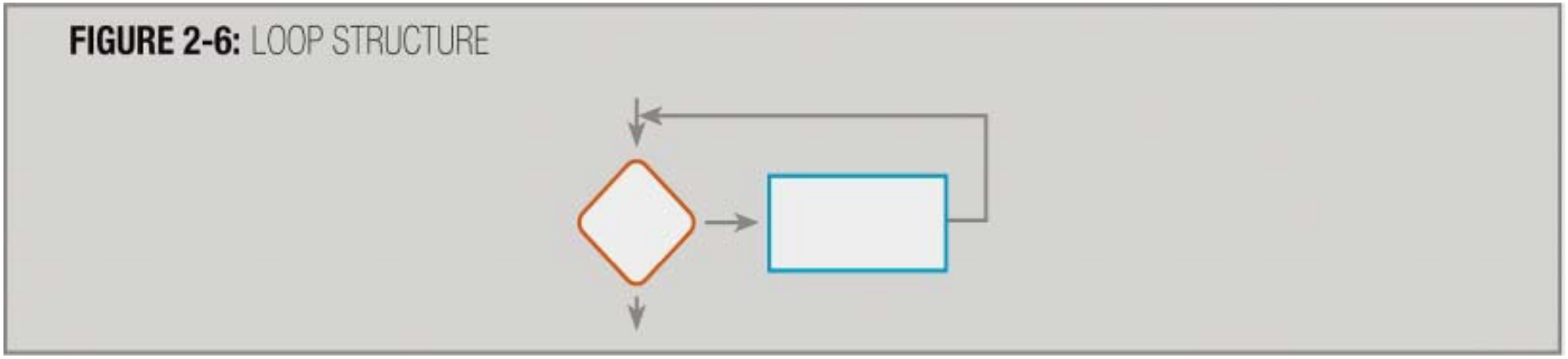
```
If varGender == "F" then
        ShowWomensClothing
End if
```

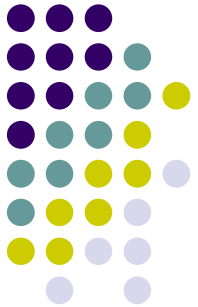
```
If varGender == "M" then
        ShowMensClothing
End if
```

```
If varGender == "F" then
        ShowWomensClothing
Else
        ShowMensClothing
End if
```

Understanding the Three Basic Structures (continued)

- **Loop structure**
  - Repeats a set of actions based on the answer to a question
  - Also called **repetition** or **iteration**
  - Question is asked first in the most common form of loop

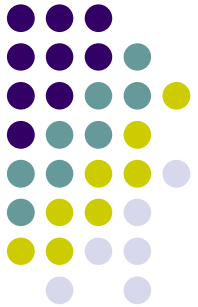




Understanding the Three Basic Structures (continued)

- **Loop structure**

```
while testCondition continues to be true
   do someProcess

while you continue to beHungry
   take anotherBiteOfFood
```



Understanding the Three Basic Structures (continued)

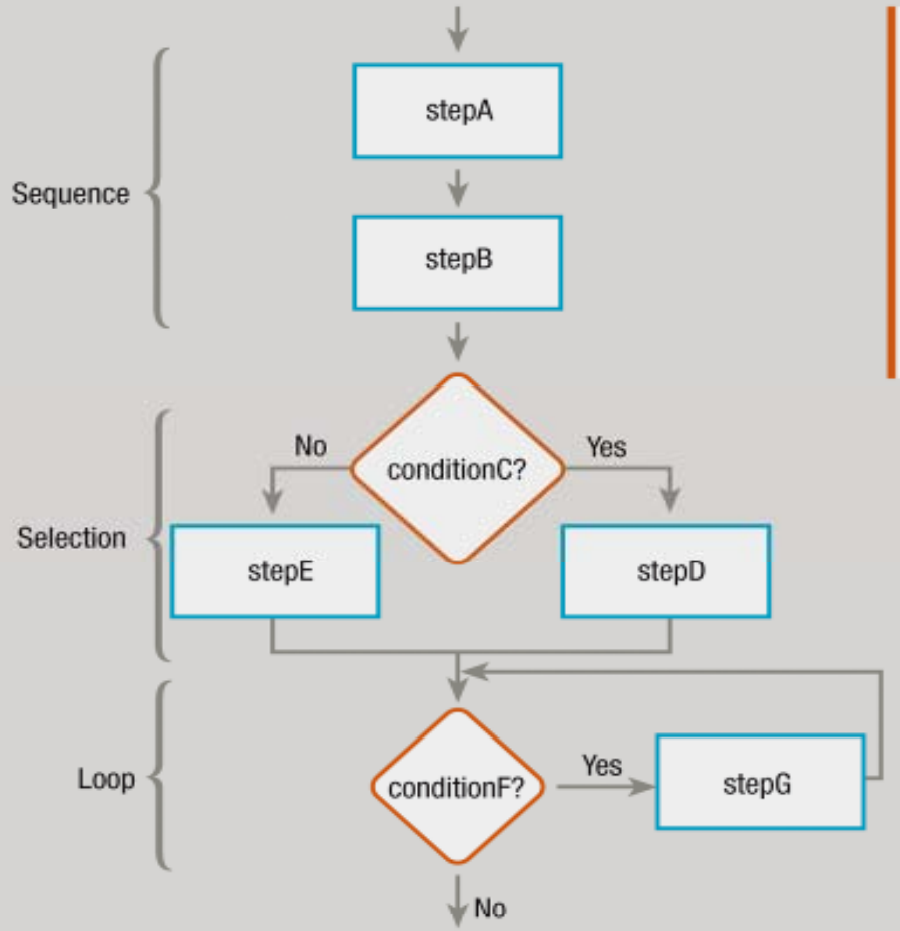
- All logic problems can be solved using only these three structures
- Structures can be combined in an infinite number of ways
- **Stacking**: attaching structures end-to-end
- End-structure statements
  - Indicate the end of a structure
  - **`endif`**: ends an if-then-else structure
  - **`endwhile`**: ends a loop structure

```
If varGender == "F" then
        ShowWomensClothing
Else
        ShowMensClothing
End if
```

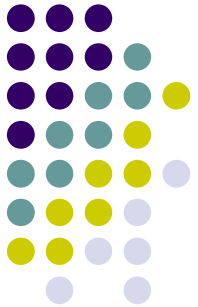
```
varCounter = 0

While varCounter <= 20 do
    Print varCounter
    varCounter = varCounter + 1
endwhile
```

Understanding the Three Basic Structures (continued)

FIGURE 2-7: STRUCTURED FLOWCHART AND PSEUDOCODE




```
do stepA
do stepB
if conditionC is true then
     do stepD
else
     do stepE
endif
while conditionF is true
     do stepG
endwhile
```

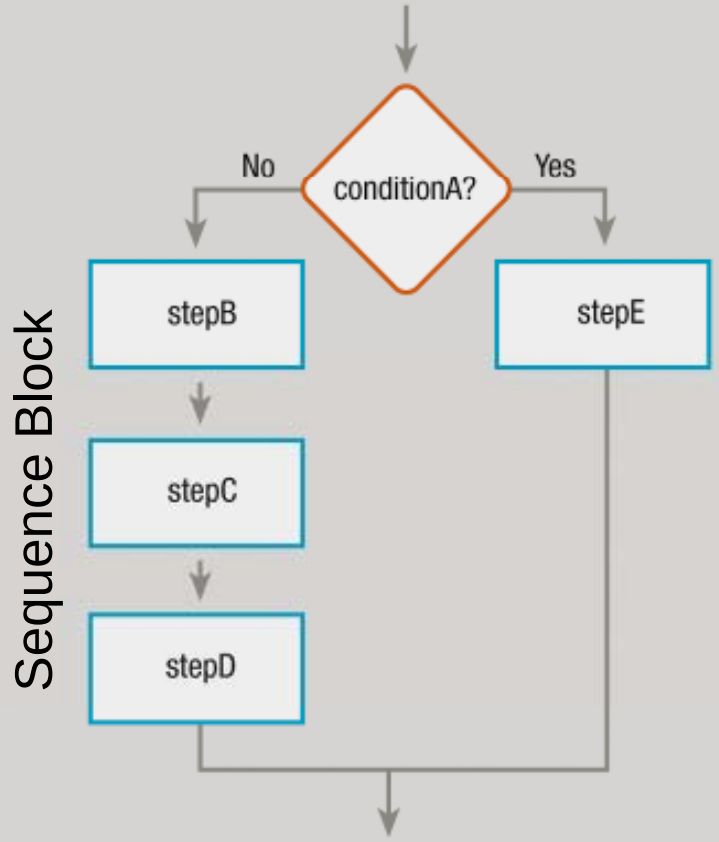


Understanding the Three Basic Structures (continued)

- Any individual task or step in a structure can be replaced by a structure
- **Nesting**: placing one structure within another
- Indent the nested structure's statements
- **Block**: group of statements that execute as a single unit

Understanding the Three Basic Structures (continued)

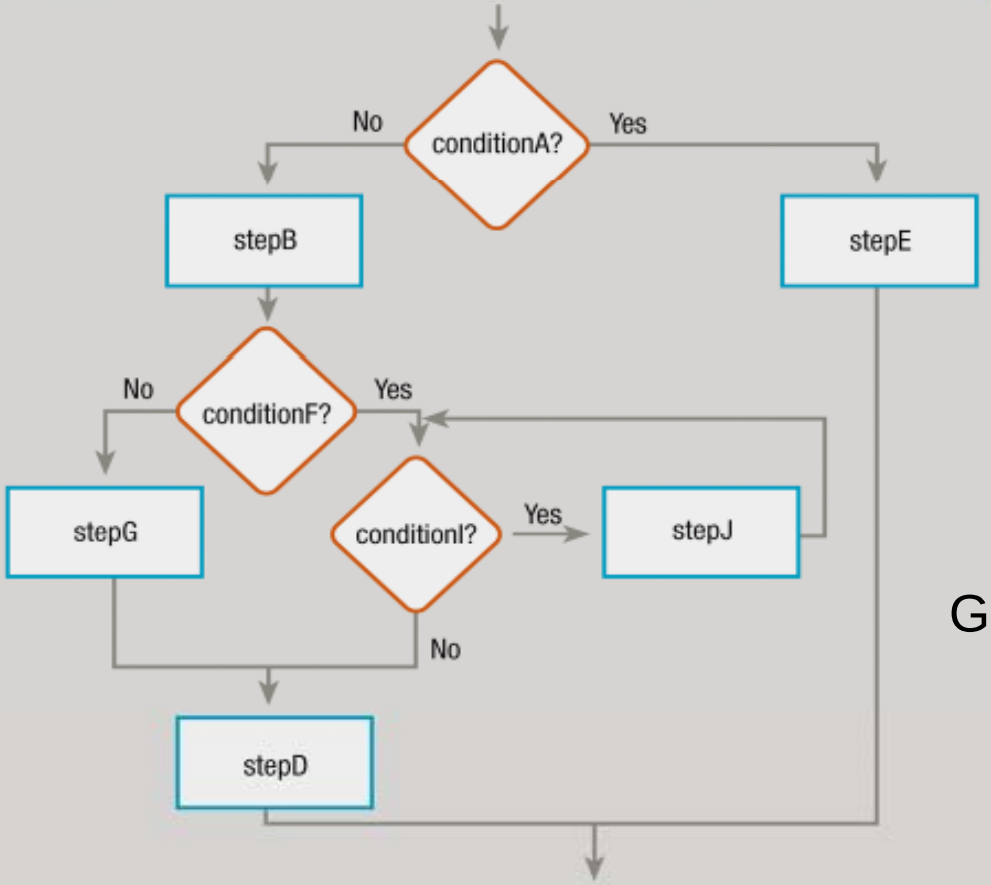
FIGURE 2-8: FLOWCHART AND PSEUDOCODE SHOWING A SEQUENCE NESTED WITHIN A SELECTION




```
if conditionA is true then
     do stepE
else
     do stepB
     do stepC
     do stepD
endif
```

Understanding the Three Basic Structures (continued)

FIGURE 2-10: FLOWCHART AND PSEUDOCODE FOR LOOP WITHIN SELECTION WITHIN SEQUENCE WITHIN SELECTION

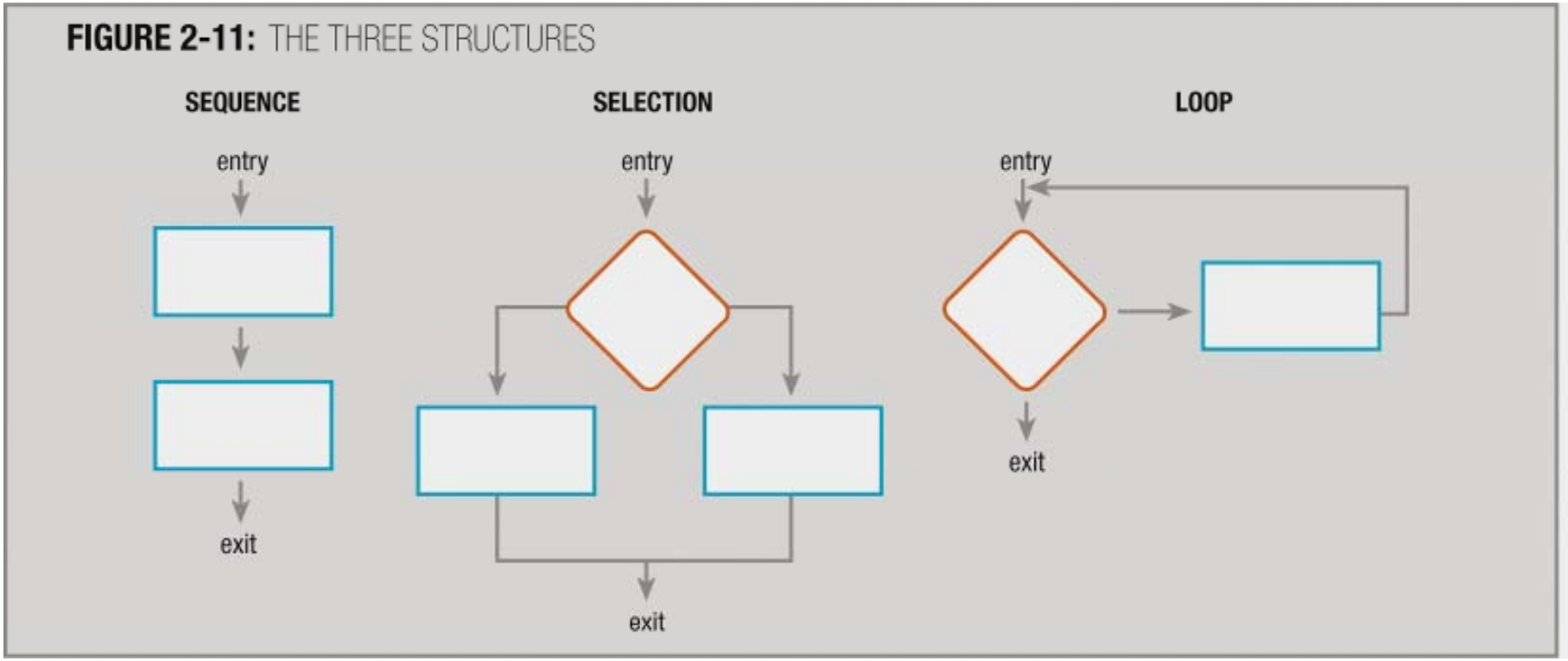


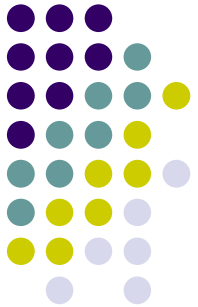

```
if conditionA is true then
     do stepE
else
     do stepB
     if conditionF is true then
          while conditionI is true
               do stepJ
          endwhile
     else
          do stepG
     endif
     do stepD
endif
```

Gluing the structures together

Understanding the Three Basic Structures (continued)

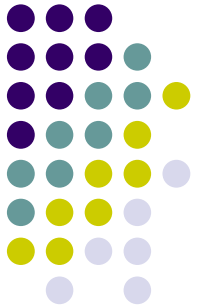
- Each structure has one entry and one exit point
- Structures attach to others only at entry or exit points
- Most initial coding issues arise from these entry and exit points not being correctly defined or receiving correct (expected) input





Using the Priming Read

- **Priming read** (or **priming input**):
  - Reads the first input data record
  - Outside the loop that reads the rest of the records
  - Helps keep the program structured
- Analyze a program flow for structure one step at a time
- Never try and code two sequential structures at once, as finding points of error becomes exponentially more difficult
- Watch for unstructured loops that do ***not*** follow this order:
  1. First ask a question
  2. Take action based on the answer
  3. Return to ask the question again



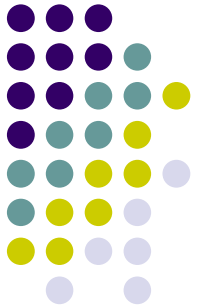
Using the Priming Read (continued)

- Unstructured loop:

```
While varCounter <= 20 do
    Print varCounter
    varCounter = varCounter + 1
endwhile
```

- Structured loop

```
varCounter = 0

While varCounter <= 20 do
    Print varCounter
    varCounter = varCounter + 1
endwhile
```



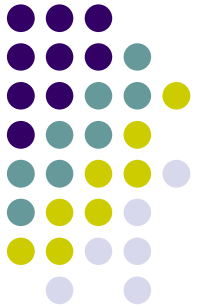
Using the Priming Read (continued)

- Structured but nonfunctional loop

```
varCounter = 0

While varCounter <= 20 do
    Print varCounter
    Endwhile
varCounter = varCounter + 1
```

- Structured, working, but non useful loop

```
varCounter = 0

While varCounter <= 20 do
        varCounter = varCounter + 1
Endwhile
Print varCounter
```

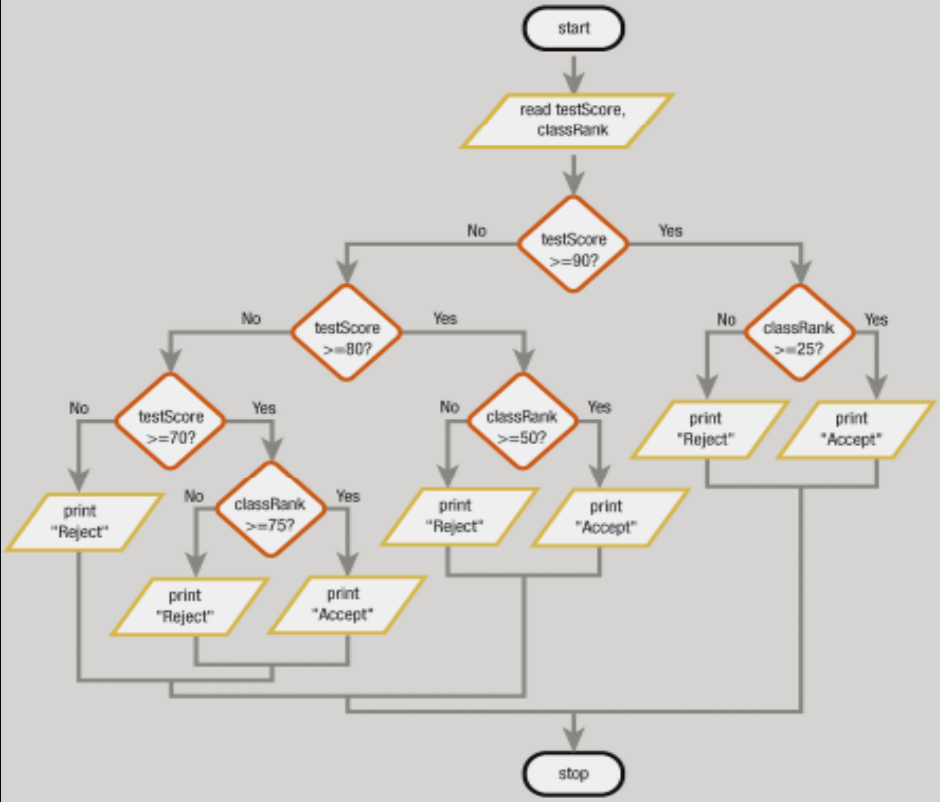


Understanding the Reasons for Structure

- Advantages of structure:
  - Provides clarity
  - Professionalism
  - Efficiency
  - Ease of maintenance
  - Supports modularity

Understanding the Reasons for Structure (continued)

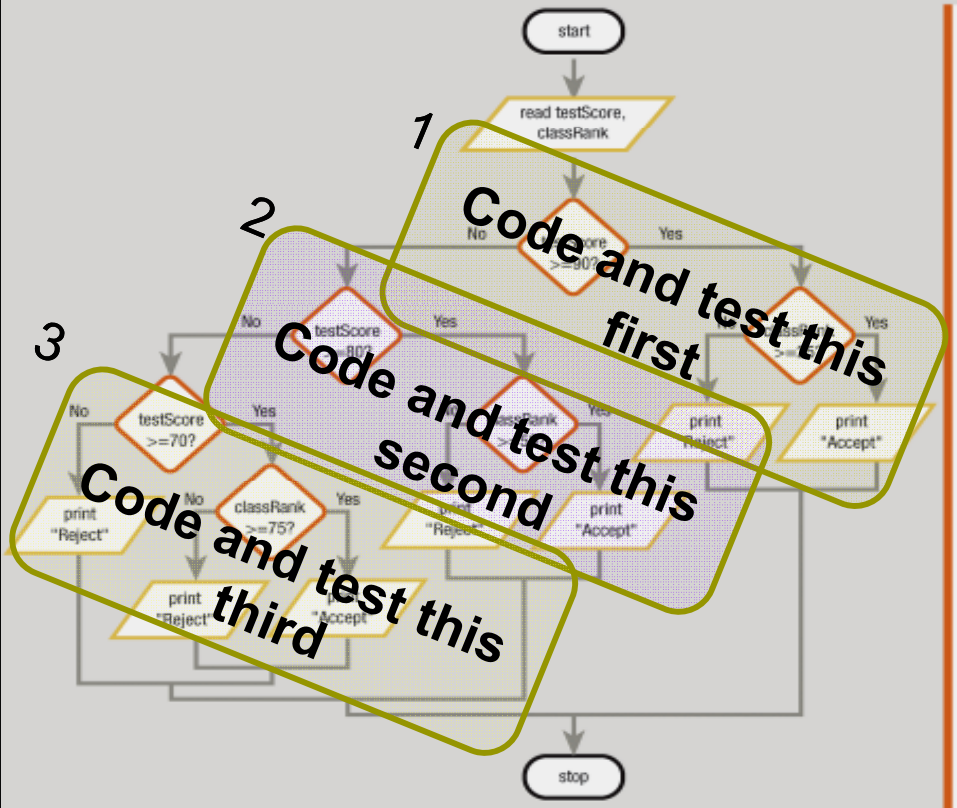
FIGURE 2-19: FLOWCHART AND PSEUDOCODE OF STRUCTURED COLLEGE ADMISSION PROGRAM



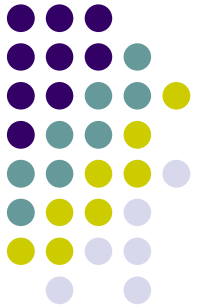

```
start
     read testScore, classRank
     if testScore >= 90 then
          if classRank >= 25 then
               print "Accept"
          else
               print "Reject"
          endif
     else
          if testScore >= 80 then
               if classRank >= 50 then
                    print "Accept"
               else
                    print "Reject"
               endif
          else
               if testScore >= 70 then
                    if classRank >= 75 then
                         print "Accept"
                    else
                         print "Reject"
                    endif
               else
                    print "Reject"
               endif
          endif
     endif
stop
```

Understanding the Reasons for Structure (continued)

FIGURE 2-19: FLOWCHART AND PSEUDOCODE OF STRUCTURED COLLEGE ADMISSION PROGRAM




```
start
     read testScore, classRank
     if testScore >= 90 then
1         if classRank >= 25 then
               print "Accept"
          else
               print "Reject"
          endif
     else
2         if testScore >= 80 then
               if classRank >= 50 then
                    print "Accept"
               else
                    print "Reject"
               endif
          else
               if testScore >= 70 then
3                   if classRank >= 75 then
                         print "Accept"
                    else
                         print "Reject"
                    endif
               else
                    print "Reject"
               endif
          endif
     endif
stop
```



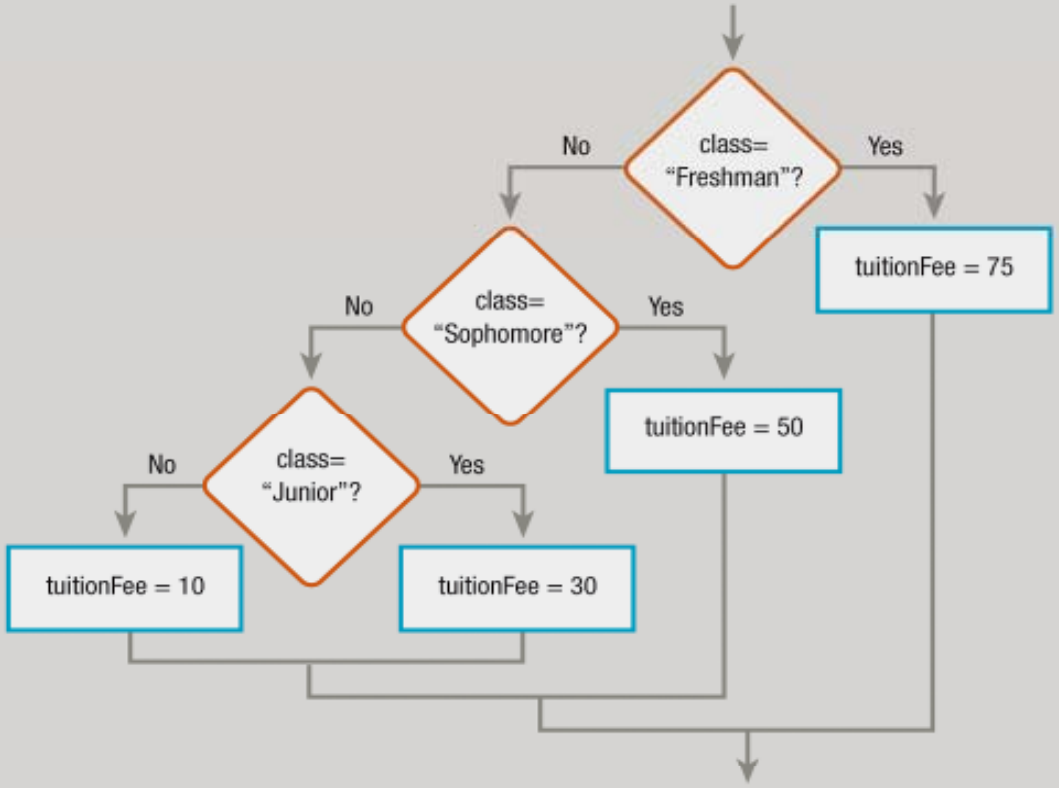
Three Special Structures – `Case`, `Do While`, and `Do Until`

- Many languages allow three additional structures:
  - **`case`** structure
  - **`do-while`** structure
  - **`do-until`** structure
- **`Case`** Structure:
  - Decisions with more than two alternatives
  - Tests a variable against a series of values and takes action based on a match
  - Nested **`if-then-else`** statements will do what a **`case`** structure does

Three Special Structures – `Case`, `Do While`, and `Do Until` (continued)

- Using nested **`if-then-else`** for multiple alternatives

FIGURE 2-31: FLOWCHART AND PSEUDOCODE OF TUITION DECISIONS

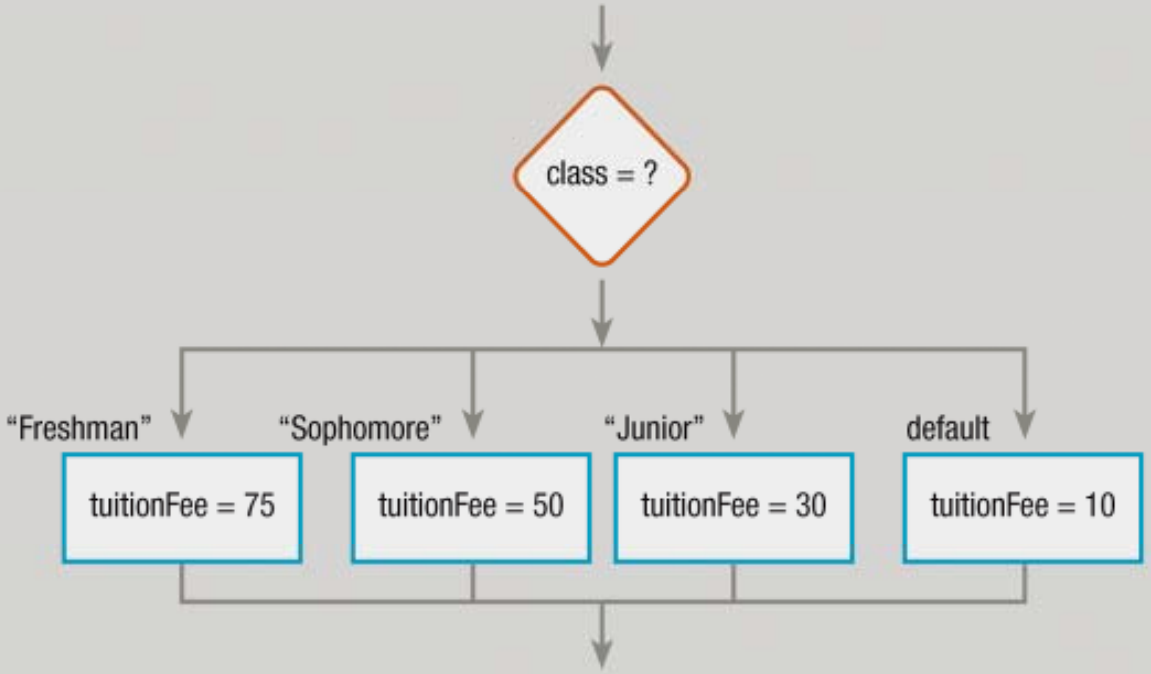



```
if class = "Freshman" then
     tuitionFee = 75
else
     if class = "Sophomore" then
          tuitionFee = 50
     else
          if class = "Junior" then
               tuitionFee = 30
          else
               tuitionFee = 10
          endif
     endif
endif
```

Three Special Structures – `Case`, `Do While`, and `Do Until` (continued)

- Using a **`case`** structure for multiple alternatives

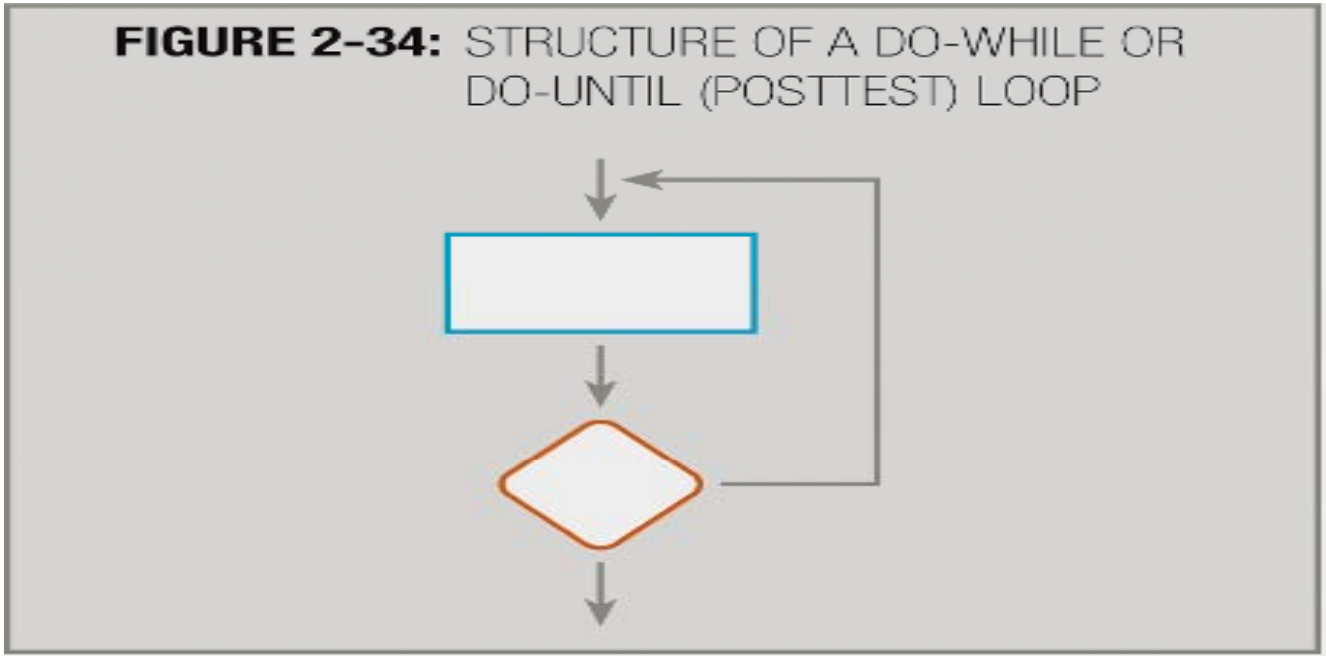
FIGURE 2-32: FLOWCHART AND PSEUDOCODE OF CASE STRUCTURE

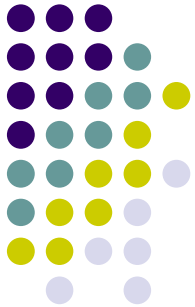



```
case based on class
     case "Freshman"
          tuitionFee = 75
     case "Sophomore"
          tuitionFee = 50
     case "Junior"
          tuitionFee = 30
     default
          tuitionFee = 10
endcase
```

Three Special Structures – `Case`, `Do While`, and `Do Until` (continued)

- **`do-while`** and **`do-until`** loops
  - Question is asked at the end of the loop structure
  - Ensures that the loop statements are always used at least once
  - Especially common in database recordset loops where you do-until there are no more records in the result set





Three Special Structures – `Case`, `Do While`, and `Do Until` (continued)

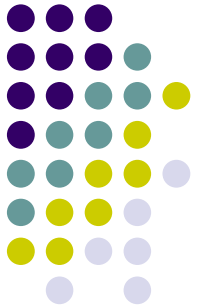
- **`do-while`** loop executes as long as the question's answer is Yes or True
- **`do-until`** loop executes as long as the question's answer is No or False (until it becomes Yes or True)

```
varCounter = 0

While varCounter <= 20 do
Print varCounter
        varCounter = varCounter + 1
Endwhile
```

Is varCounter still less than or equal to 20 – Yes? – ok, keep going

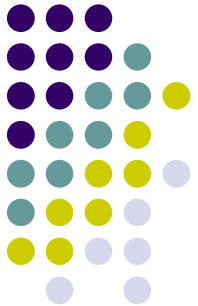
```
varCounter = 0

Do until varCounter = 20
Print varCounter
        varCounter = varCounter + 1
Endwhile
```

Is varCounter equal to 20 yet – No? – ok, keep going



Selection of coding structures

- In the end, as a developer you select the structures that you feel best suite coding tasks at hand
- Be consistent at all times
- Predictable and consistent coding structures are easier to find, debug or enhance
- As stated, in a branching conditional structure, always try and code the IF side first, using known inputs, then code the ELSE side



Setting up your priming values

- In some applications, when you commence a looping structure, you will know a certain value
- For example, you might write a program that summarises yearly sales figures for a shop that is open 24/7

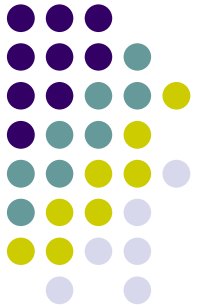
```
varYearlyTotal = 0
varDailyTake = 0

For i = 1 to 365 do
        varYearlyTake = varYearlyTake + varDailyTake(i)
End
```

- In such a case you know the number of days in a year and so it can be hard-coded into the loop

Setting up your priming values (continued)

- In many applications you will need to set up looping structures based on highly variable types of input (where you cannot hard-code a value with certainty)
- In the example of the form shown here we might end up with any number of fields to process in the unit details section of the form (assuming the user can add new rows as they go)
- In this case, we need to read in the submitted form, analyse its contents, and then setup our loop

Firstname			
Surname			
Student ID			
Course Type	Undergraduate Degree		
Unit Code	***Credit Points***	***Year / Semester***	***Mark (/ 100)***

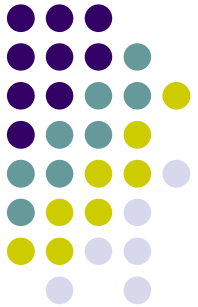


Analysing input for priming reads

- One possible approach for the previous example might be to use a for-each structure first

```
varFieldCount = 0
For Each Item in Form
        varFieldCount = varFieldCount + 1
Next

'now to process the form

For i = 1 to varFieldCount do
        'read and process input
End
```

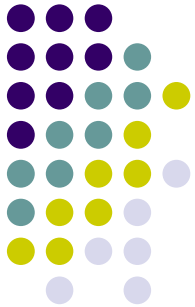
- Now, that will get the entire form and not just the unit details, so we have to go a bit further (ie eliminate the Firstname, Surname, Student ID and Course Type fields)



Analysing input for priming reads

```
varFieldCount = 0
For Each Item in Form
        If  Item.Name <> “FirstName” OR Item.Name <> “Surname OR
        Item.Name <> “StudentID” OR Item.Name <> “CourseType” then
                varFieldCount = varFieldCount + 1
        End If
Next

‘now to process the form, taking into account four fields per row
varTotalRows = varFieldCount / 4

For i = 1 to varTotalRows do
        ‘read and process input
End
```

- Now, this example is broken into rows as we would probably want to do both row and field based input processing and validation
- What would be the first thing we would check for at the row level before processing any further input?



Readings

- How to Write Unmaintainable Code by Roedy Green – Available at http://freeworld.thc.org/root/phun/unmaintain.html

  (NOTE: this is meant to be funny, but serves as good guide as to how NOT to write your code. If you find yourself doing these kinds of things, STOP! ☺