Lecture 11
**Nested conditional statements**

- A conditional statement (i.e., an *if-statement* or an *if-else-statement*) is also a statement.
- We can use an *if-statement* or an *if-else-statement* in the *then-part* (and in the *else-part*) of a conditional statement !!!
- *Nested* conditional statement = a conditional statement where the *then-part* and/or the *else-part* contains another conditional statement.
- You can **nest** conditional statement **arbitrarily deep**
- Obviously, **deeper nesting** makes the **program** **difficult to understand**
- You should:
  - Use **nested conditional statements** only when **necessary**
  - Try **not** to **nest conditional statements too deeply**
Programming example: determine the price for a hair cut

- Hair cut pricing of a saloon:
  - Male customer:
    - Boys (age \(\leq 13\)): $10
    - Men (age \(> 13\)): $15
  - Female customer:
    - Girls (age \(\leq 13\)): $12
    - Women (age \(> 13\)): $25

Write a program that:

- Reads in the \textit{sex} and the \textit{age}
- Prints the \textit{price} of the hair cut.
**Algorithm:**

```
Read in sex, age;

<table>
<thead>
<tr>
<th>sex == 'M'</th>
<th>age &lt;= 13</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>age &lt;= 13</td>
<td>false</td>
</tr>
<tr>
<td>(boy)</td>
<td>(man)</td>
<td>(girl)</td>
</tr>
<tr>
<td>price = 10;</td>
<td>price = 15;</td>
<td>price = 12;</td>
</tr>
<tr>
<td>(woman)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>price = 25;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Print price;
```
Suppose the user enters: `sex = 'M'` and `age = 11`

<table>
<thead>
<tr>
<th>sex == 'M'</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>age &lt;= 13</td>
<td>true</td>
</tr>
<tr>
<td>(boy)</td>
<td>price = 10;</td>
</tr>
<tr>
<td>(man)</td>
<td>price = 15;</td>
</tr>
<tr>
<td>(girl)</td>
<td>price = 12;</td>
</tr>
<tr>
<td>(woman)</td>
<td>price = 25;</td>
</tr>
</tbody>
</table>

Print price;
Another example: path taken by program when input is sex = 'F' (female) and age = 25

<table>
<thead>
<tr>
<th>sex == 'M'</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>age &lt;= 13</td>
<td>age &lt;= 13</td>
</tr>
<tr>
<td>(boy) price = 10;</td>
<td>(man) price = 15;</td>
</tr>
<tr>
<td></td>
<td>(girl) price = 12;</td>
</tr>
<tr>
<td></td>
<td>(woman) price = 25;</td>
</tr>
</tbody>
</table>

Print price;
Programming example: assign a letter grade to a number grade

○ Letter grade assignment:

- grade ≥ 90: A
- 80 ≤ grade < 90: B
- 70 ≤ grade < 80: C
- 60 ≤ grade < 70: D
- grade < 60: F

Write a program that:

- Reads in a number grade
- Prints the letter grade.
Algorithm using *non-nested* if-statements:

```java
import java.util.Scanner;

public class Grade01 {
    public static void main(String[] args) {
        double ng;
        String lg = "";

        Scanner in = new Scanner(System.in); // Construct Scanner object
        System.out.print("Enter number grade: ");
        ng = in.nextDouble(); // Read in next number into ng

        if (ng >= 90 )
            lg = "A";
        else if (80 <= ng && ng < 90 )
            lg = "B";
        else if (70 <= ng && ng < 80 )
            lg = "C";
        else if (60 <= ng && ng < 70 )
            lg = "D";
        else if ( ng < 60 )
            lg = "F";

        System.out.println("Letter grade = " + lg);
    }
}
```
Nested if-else

```java
public class Grade02 {
    public static void main(String[] args) {
        double ng;
        String lg = "";

        Scanner in = new Scanner(System.in); // Construct Scanner object
        ng = in.nextDouble(); // Read in next number into ng

        if ( ng >= 90 )
            lg = "A";
        else
            {
                if ( ng >= 80 )
                    lg = "B";
                else
                    {
                        if ( ng >= 70 )
                            lg = "C";
                        else
                            {
                                if ( ng >= 60 )
                                    lg = "D";
                                else
                                    lg = "F";
                            }
                    }
            }

        System.out.println("Letter grade = " + lg);
    }
}
```
Two-way selection

- A two-way selection is a choice between 2 mutually exclusive cases:

2-way selection:

- A
- not A
A three-way selection construct

3-way selection:

The choices of $A$, $B$ and $C$ are mutually exclusive.
A common way to ensure that the 3 choices are mutually exclusive is as follows:

3-way selection:

- X
- Y
- not Y

(not X)
The 3-way selection construct can be implemented using the following nested if-else statements:

```
<table>
<thead>
<tr>
<th>ConditionX</th>
<th>ConditionY</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
```

Notes:
- The statement(s) $s_1;$ are executed only when $\text{condition1} = \text{true}$
- The statement(s) $s_2;$ are executed only when $(\text{condition1} = \text{false} \text{ and } \text{condition2} = \text{true})$
- The statement(s) $s_3;$ are executed only when $(\text{condition1} = \text{false} \text{ and } \text{condition2} = \text{false})$

Each of the conditions are mutually exclusive.
○ The **3-way selection** written in **Java**:

```java
if ( conditionX )
{
    statements executed only when:
    conditionX = true
}
else
{
    if ( conditionY )
    {
        statements executed only when:
        conditionX = false
        and conditionY = true
    }
    else
    {
        statements executed only when:
        conditionX = false
        and conditionY = false
    }
}
```

Since there is one statement in the else-part, we do not need to use a block.
The **popular way** to write a **3-way selection** in Java:

```java
if ( conditionX )
{
    statements executed only when:
        conditionX = true
}
else if ( conditionY )
{
    statements executed only when:
        conditionX = false
        and conditionY = true
}
else
{
    statements executed only when:
        conditionX = false
        and conditionY = false
}
```

One statement!!
• The *N-way selection* construct
  
  • The *3-way selection* construct can be *generalized* into an *N-way selection*.

  A *N-way selection* construct looks like the following:

  ```
  if ( condition_1 )
    { 
      S_1; (one or more statements)
    }
  else if ( condition_2 )
    { 
      S_2; (one or more statements)
    }
  else if ( condition_3 )
    { 
      S_3; (one or more statements)
    }
  ... 
  else if ( condition_{N-1} )
    { 
      S_{N-1}; (one or more statements)
    }
  else
    { 
      S_N; (one or more statements)
    }
  ```

  Notes:

  • *S_1* will be executed (only) when *condition_1 = true*
  
  • *S_2* will be executed (only) when (*condition_1 = false and condition_2 = true*)
  
  • *S_3* will be executed (only) when (*condition_1 = false, condition_2 = false and condition_3 = true*)
  
  • And so on...
What is wrong in the following code?

```python
if (score >= 60.0)
    grade = 'D';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 90.0)
    grade = 'A';
else
    grade = 'F';
```
The *dangling-else* ambiguity

- **Ambiguous syntax**
  - A language allows *ambiguous syntax* is the *same sentence* can be read in *more than one way*

- **Example:** *English* (in fact, all human languages)
  - An *American History professor*
  - A *History professor* who is an *American citizen* (i.e.: *American History professor*)
  - A *professor* of *American History* (i.e.: *American History professor*)

- **How humans deal with ambiguity:**
  - Humans can *often* use *additional information* to *resolve* the *ambiguity*
    (E.g., we have additional information about the specific professor)
  - If we *cannot resolve the ambiguity*, we (humans) can *ask further questions* to resolve it.
Ambiguity in *programming languages*

- **Difference** between *natural language* and *programming language*:
  - A *natural language* *evolves* through *usage*
    - It's *evolution* is *not controlled*
  - A *programming language* is *designed* by one or a group of humans
    - It's *design* is *completely controlled*

- **Commonly held goal** in the *design* of a *programming language*:
  - A *programming language* should have an *unambiguous syntax*
  - In other words:
    - There is *exactly one way* to interpret each program statement *syntactically*
Resolving the dangling-else ambiguity syntactically:

- The keyword else is *associated* to the *nearest* keyword if that makes a syntactically correct statement.

Programming problem: *shipping cost*

- **Country code for US = 1**
  Cost to ship to a package to a destination in the US = $5.00, except for *Hawaii* which cost $10.00
  The *state code* for *Hawaii* = 50

- The **shipping cost** for a destination outside the US is $20.00

- Write a *Java program* than reads in:
  - A *country code* (integer)
  - A *state code* (integer)

  and prints the **shipping cost**
Algorithm:

<table>
<thead>
<tr>
<th>Input: country_code, state_code;</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost = 5.0; (let’s assume it’s in US)</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td><strong>true</strong></td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>country_code == 1</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td><strong>true</strong></td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>state_code == 50</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>(it’s Hawaii)</td>
</tr>
<tr>
<td>cost = 10.0;</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>Print cost;</td>
</tr>
</tbody>
</table>
There is **no ambiguity** in the **algorithm** given as a **structure diagram**

The **ambiguity** will be **introduced** when we **write** the **algorithm** in the **Java programming language** !!!

```java
    cost = 5.0;
    if ( country_code == 1 )
        if ( state_code == 50 )
            cost = 10.0;    // Hawaii
    else
        cost = 20.0;    // Outside US
```

(The algorithm is **unnecessarily** confusing because I want to **show the ambiguous syntax**....)
There are 2 different yet syntactically correct ways to read the if-statements:

```java
if (country_code == 1)
    if (state_code == 50)
        cost = 10.0;
    else
        cost = 20.0;
else
    cost = 20.0;
```

```java
if (country_code == 1)
    if (state_code == 50)
        cost = 10.0;
    else
        cost = 20.0;
```

You write this:

```
if ( country_code == 1 )
    if ( state_code == 50 )
        cost = 10.0;
    else
        cost = 20.0;
else
    cost = 20.0;
```

Can be interpreted like this:

```
if ( country_code == 1 )
    if ( state_code == 50 )
        cost = 10.0;
else
    cost = 20.0;
```

or like this:

```
if ( country_code == 1 )
    if ( state_code == 50 )
        cost = 10.0;
else
    cost = 20.0;
```

Apply "nearest-if" rule:

```
if ( country_code == 1 )
    if ( state_code == 50 )
        cost = 10.0;
else
    cost = 20.0;
```

This is how the Java compiler will "read" the statement!
Sample execution:

Enter country code: 1  (code for US)
Enter state code: 40   (not Hawaii)
Shipping cost = 20.0   (should be $5 !)

The reason is that the Java program is executed as follows:

Prints: 20 !!!
Writing *unambiguous* computer programs

- It is *very easy* to avoid the *dangling-else* ambiguity:

  ![Always use statement blocks in the then-part and else-part of conditional statements](image)

- Example:

```cpp
if ( country_code == 1 )
{
  if ( state_code == 50 )
  {
    cost = 10.0; // Hawaii
  }
}
else
{
  cost = 20.0; // Outside US
}
```
The switch statement: an \textit{N-way} selection statement

Syntax of the {	t switch-statement}:

\begin{verbatim}
switch ( INT-EXPRESSION )
{
    case INT-VALUE\textsubscript{1}: STATEMENT\textsubscript{11};
    STATEMENT\textsubscript{12};
    ...
    break;    // Marks the end of case INT-VALUE\textsubscript{1}

    case INT-VALUE\textsubscript{2}: STATEMENT\textsubscript{21};
    STATEMENT\textsubscript{22};
    ...
    break;    // Marks the end of case INT-VALUE\textsubscript{2}

    ... (more cases if desired) ...

    [default: STATEMENT\textsubscript{d1};    // Optional clause
     STATEMENT\textsubscript{d2};
     ...
     break;]
}
\end{verbatim}
• The **INT-EXPRESSION** in the **switch statement** is first evaluated

• It **INT-EXPRESSION** *must* be an **integer valued expression**

• The **result** of the **INT-EXPRESSION** is compared to the **integer values** given in the **case clauses**

• If the **result** of the **INT-EXPRESSION** is **equal** to some **INT-VALUE** in a case clause, the statements following the case clause upto the **break** statement are **executed**

• If the **result** of the **INT-EXPRESSION** is **not equal** to any **INT-VALUE** in the case clauses, then the statements in the **default case** are executed *if* it is specified *If the default case is not specified*, then the **switch statement** will terminate without executing any statement.
- **Note:** the execution of the switch statement is terminated by:
  - A `break;` statement, or
  - When *execution* reaches the end of the switch statement

- **Programming example:** translate a numeric month to a string name

```java
import java.util.Scanner;

public class Switch01 {
    public static void main(String[] args) {
        int a;
        String name = "";

        Scanner in = new Scanner(System.in); // Construct Scanner object
        System.out.print("Enter a numeric month (1-12): ");
        a = in.nextInt(); // Read in next number into a

        switch (a) {
            case 1: name = "January"; break; // For brevity, I put
            case 2: name = "February"; break; // the "break" statement
            case 3: name = "March"; break; // on the same line.
            case 4: name = "April"; break;
            case 5: name = "May"; break;
            case 6: name = "June"; break;
            case 7: name = "July"; break;
            case 8: name = "August"; break;
            case 9: name = "September"; break;
            case 10: name = "October"; break;
            case 11: name = "November"; break;
            case 12: name = "December"; break;
            default: name = "Invalid month"; break; // <-- This break statement
                // is redundant
        }
        System.out.println("Name of month = " + name);
    }
}
```
The different *integer typed* expressions allowed in the switch statement

- Previously discussed:

```java
switch ( INT-EXPRESSION )  // must be an integer valued expression
{
  ...
}
```

- The following types are *automatically converted* into an *integer typed value* in expressions:
  - byte
  - short
  - char

Therefore, we can *also* use them in the *switch statement*!!
• The long typed integer expression is *not* allowed in the switch statement

  – A *long typed integer value* is *not* automatically converted to *int*
  – Therefore, you *cannot* use a *long typed integer value* in a *switch statement* without a *casting operation*.
  – Floating point number is also not allowed.
String typed values are allowed in the switch statement in Java SE 7 and later

```java
import java.util.Scanner;

public class Switch04
{
    public static void main(String[] args)
    {
        String name = "";
        int a;

        Scanner in = new Scanner(System.in); // Construct Scanner object
        System.out.print("Enter the name of a month (lower case): ");
        name = in.next(); // Read in name

        switch (name)
        {
            case "january":    a = 1; break;
            case "february":   a = 2; break;
            case "march":      a = 3; break;
            case "april":      a = 4; break;
            case "may":        a = 5; break;
            case "june":       a = 6; break;
            case "july":       a = 7; break;
            case "august":     a = 8; break;
            case "september":  a = 9; break;
            case "october":    a = 10; break;
            case "november":   a = 11; break;
            case "december":   a = 12; break;
            default:           a = 0; break;
        }

        System.out.println("Index of month = " + a);
    }
}
```