Switch and Conditional Operator

Principles of Computer Programming I

Spring/Fall 20XX

AUGUSTA UNIVERSITY

Outline

- Switch statements
- Conditional operator

Multiple Equality Comparisons

- Situation: need to test if a variable equals one of several values
- Example: Convert a month number to its equivalent name

```
int month;
string monthName;
```

  - `month` will be a number between 1 and 12 (input)
  - If `month` is 1, `monthName` should be "January"
  - If `month` is 12, `monthName` should be "December"

Testing With If-Else-If

- One way to do it: An `else-if` statement for each possible value
- Lots of repetition: Each condition starts with `month == …`

```
if(month == 1)
{
  monthName = "January";
}
else if(month == 2)
{
  monthName = "February";
}
...
else if(month == 12)
{
  monthName = "December";
}
else
{
  monthName = "Error!";
}
```

Similar statements for values 3 to 11

Switch Statement Syntax

- Simplifies this type of comparison: one variable, multiple values

```
switch(<variable name>)
{
  case <value 1>:
    <statements>
    break;
  case <value 2>:
    <statements>
    break;
...
  default:
    <statements>
    break;
}
```

Must be a constant, not a variable

No { here

Executed if variable == value 1

Ends the case "block" of statements

Executed if variable == value 2

Note: All values must be different

Executed if variable does not equal any value

Testing with a Switch Statement

- Same result as the `if-else-if` statement, less repetition

```
switch(month)
{
  case 1:
    monthName = "January";
    break;
  case 2:
    monthName = "February";
    break;
...
  default:
    monthName = "Error!";
    break;
}
```

Variable to test: `month`

Means "`if (month == 1)`"

Means "`if (month == 2)`"

Same as the final `else`: nothing matched

Multiple Statements in a Case

- Unlike if-else-if, { } are not required for multiple lines of code

```
switch(month)
{
  case 1:
    monthName = "January";
    monthAbbrev = "Jan";
    break;
  case 2:
    monthName = "February";
    monthAbbrev = "Feb";
    break;
...
}
```

Case begins here

Case ends here

```
if(month == 1)
{
  monthName = "January";
  monthAbbrev = "Jan";
}
else if(month == 2)
{
  monthName = "February";
  monthAbbrev = "Feb";
}
...
```

break is (Usually) Required

- case statements define where code execution **starts**, not ends
- Omitting break at the end of a case is an error, like omitting a }

Error! Control cannot continue past the end of a case

```
switch(month)
{
  case 1:
    monthName = "January";
    monthAbbrev = "Jan";
    break;
  case 2:
    monthName = "February";
    monthAbbrev = "Feb";
  case 3:
    monthName = "March";
    monthAbbrev = "Mar";
    break;
...
}
```

Intentionally Omitting break

- A `case` with **no body** doesn't need a `break`
- "Combine" cases that should have the same behavior
- Regardless of which case matches, same code block executes
- Example: Initialize `season` based on month value

```
switch(month)
{
  case 1:
  case 2:
  case 3:
    season = "Winter";
    break;
  case 4:
  case 5:
  case 6:
    season = "Spring";
    break;
...
}
```

No break, not an error

Intentionally Omitting break

- Multiple `case` labels is equivalent to `||` in an `if` statement

```
if(month == 1 || month == 2 || month == 3)
{
  season = "Winter";
}
else if(month == 4 || month == 5 || month == 6)
{
  season = "Spring";
}
...
```

```
switch(month)
{
  case 1:
  case 2:
  case 3:
    season = "Winter";
    break;
  case 4:
  case 5:
  case 6:
    season = "Spring";
    break;
...
}
```

Switch Scope – A Pitfall

- All cases of a `switch` statement are in the same scope
- This means local variables must be unique to entire `switch`

```
switch(month)
{
  case 1:
    int nextMonth = 2;
    monthName = "January";
    break;
  case 2:
    int nextMonth = 3;
    monthName = "February";
    break;
...
}
```

Declare a local variable, OK so far

Error! A variable named "nextMonth" is already defined

Limitations of Switch

- Not all `if-else-if` statements can be written with `switch`
- `switch` can only test equality, not inequality/ranges

```
decimal fee = 0;
if(mileage > 1000)
{
  fee = 50.0M;
}
else if(mileage > 500)
{
  fee = 25.0M;
}
```

```
switch(mileage)
{
  case 1001:
  case 1002:
  case 1003:
  case 1004:
  ...

    fee = 50.0M;
    break;
}
```

Where would it end?
All the numbers > 1000?

Outline

- Switch statements
- **Conditional operator**

Assignment and If Statements

- Situation: Need to assign a variable based on result of condition
- Can be done with an if statement:

```
string output;
if(myInt % 2 == 0)
{
  output = "Even";
}
else
{
  output = "Odd";
}
```

Conditional Operator Assignment

- Conditional operator ? : is a shorter way to write it:

```
string output = (myInt % 2 == 0) ? "Even" : "Odd";
```

Gets value "Even" if condition is true, "Odd" if not

Condition

`true` value

`false` value

- General structure:

Must produce values of the same type

```
condition ? true_expression : false_expression;
```

Result: one of the two values

Anything that produces a `bool`

Evaluated if condition is true

Evaluated if condition is false

Conditional Operator Examples

- Compute a different expression if the number is even or odd:

```
int answer = (myInt % 2 == 0) ? myInt / 2 : myInt + 1;
```

- Provide a default value if the input is invalid:

```
double userHeight = double.Parse(Console.ReadLine());
double height = (userHeight >= 0.0) ? userHeight : 0.0;
```

- Use a Boolean variable (flag) as the condition:

```
bool adult = ...;
decimal price = adult ? 5.0m : 2.5m;
```

Summary

- Switch statements
- Conditional operator