Decision Structures: Conditional Statements

Many times in programming, you want to make a decision based on the values of the variables and select different courses of action based on that.

If Statements

The first type of decision structure allows the selection of either one or two alternatives based on a condition. The one alternative version is the **If-Then statement**.

If the condition is true, then the statements in [some action] are executed.
If the condition is false, no actions are executed.

Example:
1. Shopping problem. If you purchase $50 or more, you get 10% off for your total bill.

```
double total;
total = ??  ;   // total is assigned a value (or one from input)
if (total > 50.0)
{
     total = total - (total * 0.10);
}
```

The **If-Then-Else statement**.

If the condition is true, then the statements in [*some action*] are executed.
If the condition is false, then [*some different actions*] are executed.

2, A cell phone user monthly payment problem. If a cell phone user has the data plan, the fee = 59.99; if a user does not have the data plan, the fee = 39.99.

```
boolean dataplan; //a boolean variable
dataplan = ??    // dataplan is either true or false
if (dataplan)
{
     fee = 59.99;
}
else
{
     fee = 39.99;
}
```

Conditions as Comparisons:

In the first example, the condition used a comparison operator between two values. This type of conditional statement has the form:

```
IF (number or variable) (comparison) (number or variable)
    [some actions]
ELSE
    [some different actions]
```

Comparison Operators

	operator	*Example*	*Meaning*
Equal to	== (int, double, Boolean)	EmpCode == 0	Is employee code equal to 0?
	.equals (String)	str.equals("abc")	Is str variable equal to "abc"?
Greater than	>	Hours > 40	Are the hours greater than 40?
Greater than or equal to	>=	Revenue >= Costs	Is Revenue greater than or equal to costs?
Less than	<	thisyear < lastyear	Is this year's revenue less than last year's?
Less than or equal to	<=	Total <= 1000.0	Is the total value less than or equal to 1000 (dollars)?
Not equal to	!=	Number != 0	Is the number not equal to zero?

Conditions as Booleans:

Notice that the result of computing a comparison operator on two values is always true or false, which is a boolean value. In fact any boolean value can be used as a condition, such as in our section example where we used a boolean variable.

Nested If Statements

In the actions of an If statement, there can be more If statements. These can be used to test more conditions.

Example: Suppose that there is a String variable called PayStatus that is either "Hourly" or "Salaried", and that we want to compute the Pay of an employee using a variable Hours that tells how many hours they worked that week, and the variable PayRate that tells how much per hour that they make. Salaried employees are always paid as if they worked 40 hours, no matter how many hours they actually worked. If hourly employees work more than 40 hours in a week, then they get 1.5 times their pay rate for any hours over 40.

```
if (PayStatus.equals("Hourly"))
{
        if (hours > 40)
                pay = (payrate*40) + (1.5*payrate*(hours - 40));
        else
                pay = payrate*hours;

}
else
{
```

```
            pay = payrate*40;
      }
```

More about Conditions: Boolean Operators

If you have an If statement and you want to test if two conditions are both true, you can use && (and) operator:

```
    if (PayStatus.equals("Hourly") && hours > 40) {
      . . .
```

If you have an If statement and you want to test if at least one of two conditions is true, you can use || (or) operator:

```
    if (PayStatus.equals("Hourly") || hours > 40) {
      . . .
```

Finally, if have a comparison or other boolean value and you want to test if the condition is **not** true, you can use the ! (not) operator.

```
      if ( !(PayStatus.equals("Hourly") && hours > 40)) {
      . . .
```

More about Conditions: a common pattern of multiple conditions

One common pattern of conditions is when you want to test for different ranges of the value of a variable. For example, suppose you have a variable with the calculated test scores of a student and you want to map the scores to letter grades as follows:

Scores	Letter Grades
85 – 100	A
70 – 85	B
55 – 70	C
< 55	D

Note that there is a slight ambiguity in the statement of the score ranges, since, e.g. 85 is listed as the lowest A score and also listed as the highest B score. Let's interpret this as meaning that 85 and over is an A and anything 70 and higher up to 85 is a B.

Here is one set of conditional statements to achieve this problem:

```
      int testScore = ??;
      String letterGrade = "";

      if (85 <= testScore)
            {  letterGrade = "A";  }
      if ((70 <= testScore) && (testScore < 85))
            {  letterGrade = "B";   }
      if ((55 <= testScore) && (testScore < 70))
            {  letterGrade = "C";   }
      if (testScore < 55)
            {  letterGrade = "D";   }
```

In this solution, it is important to initialize the letterGrade to something like an empty string because the compiler doesn't know if your if statements completely cover the range of test scores, so an initializing statement makes sure that the letterGrade variable is always defined to be something.

Another solution is to give a sequence of if-then-else conditionals where each if statement inside an else can assume that the previous condition is false:

```
if ((85 <= testScore))
        {  letterGrade = "A";  }
else if (70 <= testScore)
        {  letterGrade = "B";   }
else if (55 <= testScore)
        {  letterGrade = "C";   }
else if (testScore < 55)
        {  letterGrade = "D";   }
```