Chapter 14 Selection of alternatives with if

Section 14.1 Choosing alternatives: If-then-end if, If-then-else-end if

The if statement allows a programmer to get the computer to choose among alternative actions based on whether a condition is true or false. Unlike the use of conditions with while statements, which control the repetition of a series of actions, the if is a one-time selection.

while .... do.... end do and for.... do ....end do both mark the end of the sequence of actions to be repeated with an end do. In a similar way, if statements use the words if ... then ... else ... end if. What comes between the if and then is the condition specifying the selection criterion, what comes between the then and else is the series of actions that specifies the first alternative, and what comes between the else and end if specifies the sequence of actions that are the second alternative.

Even though an if chooses among its alternative only once, if statements can be embedded in the series of actions specified in for or while statement. In other words, selecting among alternatives can be one of the actions that's repeated.

<table>
<thead>
<tr>
<th>Example 14.1.1  If-then-else-end if</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>Commentary</td>
</tr>
<tr>
<td></td>
<td>We write a loop to process all the elements of a list and figure out how many were above 3.5.</td>
</tr>
<tr>
<td></td>
<td>We initialize L to be the list of grades, and &quot;gatherer&quot; variables to count how many of each type of grade there were. We then print out a listing header.</td>
</tr>
<tr>
<td></td>
<td>Each trip through the loop looks at another item of the list. If it is ≥ 3.5, then it prints it out and adds one to the counter.</td>
</tr>
<tr>
<td></td>
<td>If it is not (which means that the item is less than 3.5), then it just adds one to the other counter.</td>
</tr>
</tbody>
</table>
Grades at least 3.5:

<table>
<thead>
<tr>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.60</td>
</tr>
<tr>
<td>3.90</td>
</tr>
<tr>
<td>3.90</td>
</tr>
<tr>
<td>3.60</td>
</tr>
<tr>
<td>3.70</td>
</tr>
</tbody>
</table>

There were 5 grades at or above, and 5 grades below 3.5.

A fairly common occurrence is having the `else` alternative specify no actions at all. Rather than have `if...then...else end if`, Maple allows you to say just `if...then...end if`. After the repetition is finished, it prints out a message that includes the information accumulated during the loop.

We could have accumulated the count using the techniques involving functions and `map` that we looked at in Chapter 5. However, doing it this way allows us to do several things in one pass through the items in the list.
**Example 14.1.2  If-then-end if**

<table>
<thead>
<tr>
<th>Example</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>In University of West Dakota at Hoople, the Dean's list is determined by computing the average of all student's gpas, and selecting those whose gpa is at least one point higher than the average.</td>
<td></td>
</tr>
</tbody>
</table>
| \[ L := [3.6, 3.9, 2.1, 3.9, 2.1, 0.7, 1.3, 2.8, 3.6, 3.7, 3.8, 2.0]: \]
| \texttt{deanCount := 0;} |
| \texttt{avg := convert(L,`+`)/nops(L):} |
| \texttt{printf("Grade average of \%d students is: \%f.\n", nops(L), avg);} |
| \texttt{printf("Grades at least one point above average (Dean's List):\n");} |
| \texttt{for i from 1 to nops(L) do} |
| \texttt{x := L[i];} |
| \texttt{if (x >= (avg+1) )} |
| \texttt{then} |
| \texttt{printf("%.2f\n", x);} |
| \texttt{deanCount := deanCount+1;} |
| \texttt{end if;} |
| \texttt{end do:} |
| \texttt{printf("\n"); #Print a blank line} |
| \texttt{printf("There were \%d students making the Dean's List.",deanCount);} |

We initialize \( L \) to be the list of grades, and a "gatherer" variables to count how the number of Dean's list awardees. We could have written a loop to process the list and compute its average, but since we can do it in one line already, we don't really need to do it differently.

We then print out a listing header.

Each trip through the loop looks at another item of the list. If it is \( \geq \text{avg} + 1 \), then it prints it out and adds one to the counter.

Since there is no "else" nothing happens if the item is below the cutoff point.

After the repetition is finished, it prints out a message that includes the information accumulated during the loop.

We could have accumulated the count using the techniques involving functions and \texttt{map} that we looked at in Chapter 5. However, doing it this way allows us to do several things in one pass through the items in the list.
Section 14.2  Choosing among cases with \textit{If-then-elif-elif...else-end if}

The \textit{if} statements in Section 14.1 expressed control of two varieties:

1. The \textit{condition} was used to select between two alternative series of actions.  
2. The \textit{condition} was used to select between one series, and doing nothing as the alternative.

Another typical situation is to select between several (more than two) alternatives. Maple has a way to express this:

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{Table 14.2.1 The if-then-elif-else-end if} & \\
\hline
\textbf{General form} & \textbf{Notes} \\
\hline
\texttt{if (condition 1) then} & \textit{If condition 1} is true when the statement is executed, then the first series of statements is executed.  \\
\texttt{ statements separated by ; or :} & \\
\texttt{elif (condition 2) then} & Otherwise, if \textit{condition 1} is not true but \textit{condition 2} is true, then the second series as executed.  \\
\texttt{ statements separated by ; or :} & \\
\texttt{(more alternatives specified by elif)} & Additional conditions will be checked if none of the previous ones turn out to be true. The sequence of actions corresponding to the first true condition will be executed.  \\
\texttt{else} & \textit{If none of the conditions are true, then last series of statements are done.}  \\
\texttt{ statements separated by ; or :} & \\
\texttt{end if} & As with the if-then-end if of Section 14.1, the \textit{else} section can be omitted. This is equivalent to doing nothing if none of the other conditions apply. \\
\hline
\end{tabular}
\end{table}

One way to view this is that you are describing different cases, depending on what is true. The \textit{else} portion of the statement specifies what is to be done if none of the previous cases apply.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{Example 14.2.1 If-then-elif-elif-...-else-end if} & \\
\hline
\textbf{Example} & \textbf{Commentary} \\
\hline
\end{tabular}
\end{table}
We write a loop to process all the elements of a list of grades and figure out how many of each final letter grade there were.

Each trip through the loop looks at another item of the list. The first case we check for is if the grade is $\geq 3.6$. If so, it's an A and we add one to the count of As.

The second case is that the grade is $< 3.6$ and $\geq 2.6$. In this case, we add one to the counter for Bs.

The third and fourth cases check for Cs and Ds.

The "else" condition is if all the other conditions have failed. We know at that point that the grade is less than 1.0, so (according to our rules for letter grades) must be an F.

We define a "plural" function whose value is "s" if the count is more than one. Otherwise its value is "", the empty string (which is not NULL, the empty sequence, but rather the string with no characters in it).

We are actually executing

```
printf("There were %d A%s, %d B%s, %d C%s, %d D%s, and %d F%s. ", 5,"s", 1,"", 2, "s", etc. );
```

We could have done the printing
L := [3.6, 3.9, 2.1, 3.9, 2.1, 0.7, 1.3, 2.8, 3.6, 3.7];
countA := 0;
countB := 0;
countC := 0;
countD := 0;
countF := 0;

# Check for As (3.6 and above), Bs (2.6 and below an A), Cs (1.6 and below a B), Ds (1.0 and below a C), and Fs (below 1.0).

for i from 1 to nops(L) do
    x := L[i];
    if (x >= 3.6) then
        countA := countA +1;
    elif (x >= 2.6) then
        countB := countB +1;
    elif (x >= 1.6) then
        countC := countC +1;
    elif (x > 1.0) then
        countD := countD +1;
    else # we will be here only if everything else hasn't worked
        countF := countF +1;
    end if;
end do:

plural := (n) -> piecewise(n>=2, "s", n<2, "");
but it was shorter this way. Sometimes piecewise functions are a shorter way to do something that could be done with an if statement instead.

[3.6, 3.9, 2.1, 3.9, 2.1, 0.7, 1.3, 2.8, 3.6, 3.7]

There were 5 As, 1 B, 2 Cs, 1 D, and 1 F.

It is possible to include if statements as one of the actions inside another if statement. This is useful where a case may have subcases. We will discuss this situation in more detail in Chapter 15.

Section 14.3 For the curious: what's the difference between Ifs and piecewise expressions?

Astute readers may have noticed similarities between the operation of piecewise expressions or functions, and if statements. The evaluation of a piecewise function \( f(x) \) for a particular value for its argument \( x \), results in choosing one of the alternatives listed in the piecewise expression. Typically, the piecewise expression lists on each line a condition which, if true, causes the alternative listed on that line as the result of evaluating the expression. This is similar to an if condition then ... elif condition2 then .... elif condition3 ... else ... end if.

However, there are significant differences. Maple treats piecewise expressions as mathematical objects. One can get Maple to perform arithmetic with piecewise expressions, and math operations in the same way that it does with numbers or expressions such as \( x^2 + \sin(2 \cdot y - x) \). Operations such as such as diff or solve also work with piecewise expressions. if statements can specify a series of actions to be performed, which is much harder to do with piecewise expressions. However, one cannot easily perform mathematical operations on if statements.

Another implication of this difference is that piecewise expressions can be evaluated even if not all variables in them have values, whereas an if statement will fail in its execution if its conditions
cannot be decided upon when the statement is executed. Thus, piecewise expressions are for when you want to do mathematical computations where some of the expressions have branches, whereas if statements are more useful for controlling the execution of scripts, which may involve drawing pictures or working on character strings instead of mathematical operations.

So far, we lack the ability to create a function with an assignment, or a while, for, or if in it, so if you want to perform bulk computation on a list of values through map or sum, then you must use piecewise expressions to help create the function. However, in Chapter 16, we learn how to create functions that encapsulate scripts with while, for, and if.

### Example 14.3.1 Differences between piecewise expressions and if statements

A piecewise expression can be used in an expression even if its variables do not have any values. The result is just the expression, in the same way that the result of evaluating $x^2 + x + 3 + x$ is $x^2 + 2x + 3$ even if $x$ has no assigned value.

An if statement gives an error if you execute it with no assigned value for its variables, because Maple can't decide whether the condition is true or false at the time when the if is executed.

You can add piecewise expressions to other expressions. Maple arithmetic and operations such as solve and diff will attempt to work on a piecewise expression.

The reason why the derivative doesn't exist at $t=0$ and $t=5$ is because the pieces don't fit together smoothly, so the left limit and right limits don't agree.

<table>
<thead>
<tr>
<th>$pExpr := \begin{cases} 3\cdot t &amp; 0 \leq t &lt; 5 \ 2\cdot t &amp; t \geq 5 \ 3\cdot t &amp; 0 \leq t \text{ and } t &lt; 5 \ 2\cdot t &amp; 5 \leq t \end{cases}$</th>
<th>A piecewise expression can be used in an expression even if its variables do not have any values. The result is just the expression, in the same way that the result of evaluating $x^2 + x + 3 + x$ is $x^2 + 2x + 3$ even if $x$ has no assigned value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>if $t \geq 0 \text{ and } t &lt; 5$ then $pIf := 3\cdot t;$ elif $t \geq 5$ then $pIf := 2\cdot t;$ end if; Error, cannot determine if this expression is true or false: $0 \leq t$ and $t &lt; 5$</td>
<td>An if statement gives an error if you execute it with no assigned value for its variables, because Maple can't decide whether the condition is true or false at the time when the if is executed.</td>
</tr>
<tr>
<td>$pExpr + 2\cdot t$ (\begin{cases} 3\cdot t &amp; 0 \leq t \text{ and } t &lt; 5 \ 2\cdot t &amp; 5 \leq t \end{cases}) + $2\cdot t$</td>
<td>You can add piecewise expressions to other expressions. Maple arithmetic and operations such as solve and diff will attempt to work on a piecewise expression.</td>
</tr>
<tr>
<td>$pExpr2 := \text{simplify}(1.3.2)$ (\begin{cases} 2\cdot t &amp; t &lt; 0 \ 5\cdot t &amp; t &lt; 5 \ 4\cdot t &amp; 5 \leq t \end{cases})</td>
<td>The reason why the derivative doesn't exist at $t=0$ and $t=5$ is because the pieces don't fit together smoothly, so the left limit and right limits don't agree.</td>
</tr>
<tr>
<td>diff ($pExpr2, t);$ (\begin{cases} 2 &amp; t &lt; 0 \ \text{undefined} &amp; t = 0 \ 5 &amp; t &lt; 5 \ \text{undefined} &amp; t = 5 \ 4 &amp; 5 &lt; t \end{cases})</td>
<td></td>
</tr>
</tbody>
</table>
Math operations don't work easily with if.
Here we get an error message saying that we can't add the expression $2 \cdot t$ to an if.

Section 14.4  For the curious: comparing control statements in conventional programming languages

Almost all conventional programming languages (C, Mathematica, C#, Java, Matlab, Maple, C++, Python, Perl, Visual Basic, etc.) have the "loop" idea and the "selection/case" ideas in them. The main difference in how they specify them through the use of different words or symbols. Java for example, uses \{ and \} rather than "then" or "end if" to delimit the begin and end of segments of actions.

Example 14.4.1 Portion of a Java program with a conditional

```java
int countThree, countSubThree;
int [] L = {1,5,9,2,6,7,-2,1,7};
// Set up counters
countThree=0;
countSubThree=0;
// Loop through L counting how many are above three
for (int i=0; i<L.length; i++)
{
    if (L[i]>3)
    {
        countThree++;
    }
    else
    {
        countSubThree++;
    }
}
System.out.println("three and above:"+countThree+" below three: "+ countSubThree);
```

A Matlab program with a loop and a conditional may look similar to a Maple program, although it uses different functions to extract size ("size" rather than "nops") and there is different syntax (no "do", just "end", ";=" instead of "from" and ":" instead of "to"). For the most part these are differences that one gets used to -- the important thing is to recognize that the control structures work in basically the same way and look similar.
%Program to count how many elements of a vector/list are greater than and less than three
countThree=0;
countSubThree=0;
L=[1,5,9,2,6,7,-2,1,7];
[r,c]=size(L);

%Loop for i from 1 to c, where c is the number of items in L (length of L)
for i=1:c
    if L(i)>3
        countThree=countThree+1;
    else
        countSubThree=countSubThree+1;
    end;
end;
%Use formatted printing to print out things.
sprintf('Three and Above: %d, Below three: %d', [countThree, countSubThree])
**Example 14.4** Error messages from forgetting to put in parts of an if statement.

<table>
<thead>
<tr>
<th>Example</th>
<th>Commentary</th>
</tr>
</thead>
</table>
| x := 3;<br>countThree := 0;<br>countSubThree := 0;<br><br>if x>3<br>then<br>countThree := countThree +1;<br>else<br>countSubThree :=<br>countSubThree +1;<br>end if;<br><br>printf("three and above:%d,<br>below three: %d",
countThree, countSubThree); | No missing pieces here. |
| 3 0 0 1 three and above:0, below three: 1 | We forgot to put in the word "then". |
x := 3;
countThree := 0;
countSubThree := 0;

if x>3
  countThree := countThree +1;
else
  countSubThree :=
countSubThree +1;
end if;

printf("three and above:%d,
below three: %d",
countThree, countSubThree);

3
0
0

Error, missing operator or `;`

We forgot to end the statement with end if (or fi).
Warning, premature end of input, use <Shift> + <Enter> to avoid this message.

\begin{verbatim}
x := 3;
countThree := 0;
countSubThree := 0;

if x>3 then
countThree := countThree +1;
else
countSubThree := countSubThree +1;
end do;

printf("three and above:%d, below three: %d",
countThree, countSubThree);
\end{verbatim}

\begin{center}
3
0
0
\end{center}

\textbf{Error, reserved word `do` unexpected}

We get the same error message as in the second item in this example. However, this time the problem is that we left off the semi-colon after the end if. We need it if there's an instruction after the if statement in the script.

We said 'end do" rather than "end if".

Programmers need to the
x := 3;  
countThree := 0;  
countSubThree := 0;  

if x>3 then  
    countThree := countThree +1;  
else  
    countSubThree := countSubThree +1;  
end if  

printf("three and above:%d,  
below three: %d",  
    countThree, countSubThree);  

| 3     |
| 0     |
| 0     |

Error, missing operator or `;`.

Consider that there could be more than one kind of mistake that would provoke an error message. This kind of situation can be encountered with almost any sort of programming system (e.g. C, Matlab, Java, etc.) due the limitations of conventional language-processing technology.

1. Consider rewriting the if statement if the alternatives seem redundant. If some actions are done by all the alternatives (no matter what), consider moving them to be executed as the first action after the if statement, which will achieve the same result.
2. Don't forget that you can use or, not, and and to build more complicated conditions. You can also assign the results of conditions to a variable and use the variable in other conditions. This can simplify what you write, making it more likely that your script will work correctly, or work correctly with less debugging effort.

### Table 14.Z.1 If statement summary

<table>
<thead>
<tr>
<th>General form</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>if condition then action;</td>
<td>If condition is true when the statement is executed, then the</td>
<td></td>
</tr>
</tbody>
</table>
if \( x \geq 3.5 \)
then
  \( \text{printf}("%.2f\n", \ x); \)
  \( \text{countHigh} := \text{countHigh+1}; \)
end if;

If \( \text{condition1} \) is true when the statement is executed, then the actions are performed. If not, then \( \text{condition2} \) is true, then the second group of actions is performed, etc. If none of the conditions are true, then nothing is done.

This is like a "case by case" description of what to do, where the various conditions describe the cases.
# Leaves weather variable unchanged except for two cases where there's change.

```python
if barometer="falling" and weather="It's sunny"
    then
        weather := "It's cloudy";
    elif barometer="rising" and weather="It's cloudy"
        then
            weather="It's sunny";
    end if;
```

| if condition1 then action; .... action; elif condition2 actions2; .... actions2; elif condition3 | Like the previous situation, only there is an additional set of actions that are done if none of the conditions are true. This is like a "case by case" description of what to do, where the various conditions describe the cases, and |
actions3; .... actions3; else actions otherwise; ... actions otherwise. end if

the last series of actions describe what happens otherwise if none of the other cases hold.

# Determine the location of an (x,y) point.
# We assume that x and y already have values from
# assignments made in the previous point in execution.
if x>=0 and y>=0 then
    location := "quadrant 1";
elif x>=0 and y<0 then
    location := "quadrant 4";
elif x<0 and y>=0 then
    location := "quadrant 2";
elif x<0 and y<0 then
    location := "quadrant 3";
else
    location := "unknown";
end if;

Table 14.Z.2 When to use piecewise expressions and functions, and when to use if statements

<table>
<thead>
<tr>
<th>piecewise expressions describe mathematical alternatives that an expression can taken on.</th>
<th>if statements describe alternative series of actions to be taken.</th>
</tr>
</thead>
<tbody>
<tr>
<td>It is hard to perform an assignment using a piecewise expression or function.</td>
<td>An example of an action that can be performed with an if is an assignment, or a print/printf for a report being generated line-by-line in a loop.</td>
</tr>
<tr>
<td>Arithmetic, <code>solve, diff, int</code>, etc. work on piecewise expressions and functions</td>
<td>Arithmetic, <code>solve, diff, int</code>, etc. don't work on if statements.</td>
</tr>
</tbody>
</table>