Selection Statements

Topics:

+ If Statements
+ If-Else Statements
+ Ternary Statements
+ Else-If Statements
+ Recursive Functions

Resources:

- `if.cpp` - `ifelse.cpp` - `elseif.cpp` - `recur.cpp`

Introduction

Selection statements are the first of the control structures that we will discuss. They are statements that make decisions. This means that they execute or skip a colletion of statements based on the outcome of a *condition* (boolean argument).

A few important things to know about selection statements are:

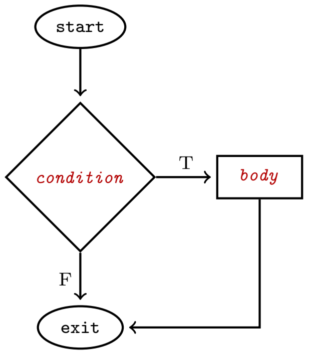
- A selection statement is a collection of special statements.
- Every statement of a selection statement has a body.
- At most one statement's body of a selection statement is executed.
- Selection statements can be nested.
- The order of the statements of a selection statement matters.
- It is possible to define a return-value functin that does not have a reachable return statement with selection statements.

If Statements

Whenever there is a situation where you need to execute a set of statements only if a condition is met, you would use a simple if statement. Its syntax is

```
if(condition)
{body}
```

It operates as follows



The flowchart is stating that when the code executes an if statement, it initially evaluates the condition of the if statement. If the condition evaluates to true, the body of the if statement is executed, and then, the rest of the code. However, if the condition evaluates to false, the body of the if statement is skipped and the rest of code is executed immediately. It is important to note, that the body of the if statement is executed at most once. For instance, the absolute value function

$$|x| = \begin{cases} -x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

can be written as the function

```
double Abs(double x)
{
 if(x < 0)
 {
  x *= -1;
 }
 return x;
}
```

```
double Abs(double x)
{
 if(x < 0)
 {
  return (-1 * x);
 }
 return x;
}
```

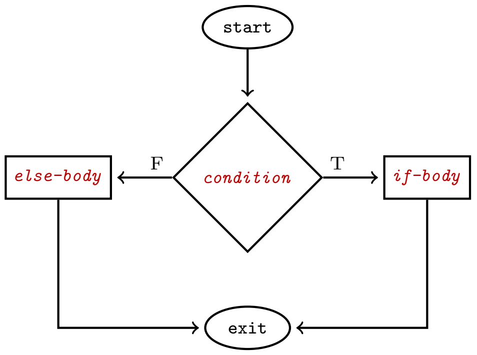
Although the definition of the absolute value function (mathematical version) has two conditions, you only need to modify the input for one of them; hence, a simple if statement will suffice. We provided two definitions to elaborate that functions can be written in multiple ways. But the differences in the definitions will only result in, at most, an addition step in traces of negative values. Last, since the body of an if statement is not guaranteed to execute, make sure there is a return statement outside of the body of the if statement; otherwise, you will receive an unreachable return error.

If-Else Statement

There are some instances where you would like to do a certain set of things when a condition is true, and another set of things when it is false. In those instances, we will use an if-else statement. Its syntax is

```
if(condition)
{body}
else
{body}
```

The if-else statement consists of an if statement that must be immediately followed by an else statement which does not every have a condition. It operates as follows



This means that when an if-else statement is being executed, the condition of of the if statement is evaluated first as usual. If it evaluates to true, the body of the if statement is executed; however, if it evaluates to false, the body of the else statement is executed. Regardless, of the outcome of the evaluation of the condition, a body of one of the statements is guaranteed to execute. Furthermore, as stated before in the introduction, only one body will every be executed.

It is significant to mention that you should **never** use an if-else statement if your objective is to only display, assign or return the outcome of a boolean argument or its negation. It is not necessarily incorrect, but it is a waste of time and amateur coding. For instances,

```
bool IsEven(int n)
{
 if(n % 2 == 0)
 {
  return true;
 }
 else
 {
  return false;
 }
}
```

```
bool IsEven(int n)
{
 return (n % 2 == 0);
}
```

```
string IsEvenOrOdd(int n)
{
 if(n % 2 == 0)
 {
  return "even";
 }
 else
 {
  return "odd";
 }
}
```

The first two functions, `IsEven()`, starting from the left are doing exactly the same thing, but the leftmost function is adding unnecessary lines by using the if-else statement. But the rightmost function is an example of the valid use of the if-else statement.

Ternary Statements

There are situations where you would need to use an if-else statement to simply assign a value to a variable or to return a value such as for the function below and for the rightmost function from the previous example.

```
void GetMin(int n,int m,int& lo)
{
 if(n <= m)
 {
  lo = n;
 }
 else
 {
  lo = m;
 }
}
```

These statements can be simplified with the use of a *ternary statement* which has the following syntax

`(condition)?argument:argument`

where the arguments **must be the same type**. The condition of the ternary statement is evaluated initially. If it evaluates to true, the first argument (left) is executed; otherwise, the second argument (right) is executed. To further clarify, the following functions are rewrites of the functions `IsEvenOrOdd()` and `GetMin()` that implement the ternary statement.

```
string IsEvenOrOdd(int n)
{
 return (n % 2 == 0)?("even"):(""odd"");
}
```

```
void GetMin(int n,int m,int& lo)
{
 lo = (n <= m)?(n):(m);
}
```

It is considerable time saver and space reducer. In the examples above the arguments are in paratheses. This practice helps distinguish the arguments especially when they are long expressions. However, although using ternary statements make coding easier, do not use them if the readability of your code will be significantly compromised such as in the following code

```
char Base(int n)
{
 return (n % 2 == 0)?((n % 4 == 0)?('a'):('t')):((n % 4 == 1)?('c'):('g'));
}
```

which is equivalent to

```
char Base(int n)
{
 if(n % 2 == 0)
 {
  return (n % 4 == 0)?('a'):('t');
 }
 else
 {
  return (n % 4 == 1)?('c'):('g');
 }
}
```

The first version is shorter, but you are more likely to make a mistake and it may be harder to understand.

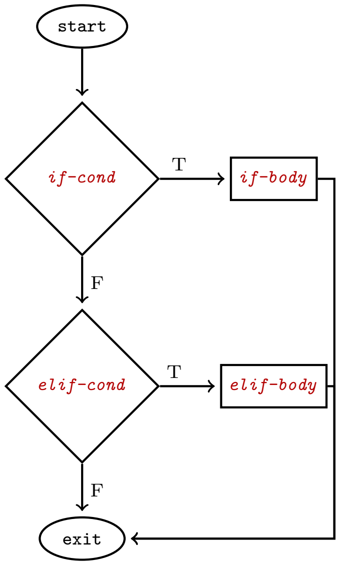
Else-If Statements

There are situations where there are more than a condition and its negation such as determining a letter grade, or determining if a integer is positive, negative or zero. In these situations, you will use an else-if statement. Its syntax is

```
if(condition)
{body}
[ else if(condition) ]+
[ {body}             ]
```

An else-if selection statement can have 1 or more else-if statements immediately following an if statement. Each else-if statement is written as the keyword else followed by the keyword if with a condition in parathesese, and then, a body.

To illustrate how it operates, the following flowchart provides the scenario with a single else-if statement



It implies that the condition of the if statement is evaluated, first. If the condition evaluates to true, the body of the if statement is executed, and then, the remainder of the selection statement is skipped and the rest of the code will be executed. However, if the condition of the if statement is false, the condition of the else-if statement is evaluated. If it evaluates to true, the body of the else-statement will be executed, and then, rest of the code will be executed; otherwise, just the rest of the code will be executed.

If there were more else-if statements, the condition of the next else-if statement will be evaluated if the first else-if statement evaluated to false, and so on, until either all conditions evaluate to false and the selection statement terminates, or a condition evaluates to true and the body of the else-if statement of that condition is executed. Once one condition evaluates to true, none of the following else-if statements will be visited, even if their conditions may evaluate to true as well. But recall, as stated in the introduction, at most one body will be executed.

To elaborate, consider the following function

```
char LetterGrade(double grd)
{
 if(grd >= 90)
 {
  return 'A';
 }
 else if(grd >= 80)
 {
  return 'B';
 }
 else if(grd >= 65)
 {
  return 'C';
 }
 else if(grd < 65)
 {
  return 'F';
 }
}
```

If `grd` were 89 both the second and third conditions will evaulate to true; however, the body of the statement of the second condition will be executed because it will be the first condition to evaluate true. This means that the conditions do not have to be mutually exclusive.

Last, although the selection statements are introduced as different statements, they are parts of a whole, the if statement. Hence, the syntax of a full if statement is

```
 if(condition)
 {body}
[ else if(condition) ]*
[ {body}             ]
[ else   ]?
[ {body} ]
```

The if statement must start the selection statement, the else statement must end it if an else statement is included and the else-if statements are in between.