Lecture 1: Selection Statements

Introduction

Control structures allow code to make decisions; meaning they determine which statements to execute. The first type of control structure is a selection statement. A selection statement executes or skips a set of statements based on a condition evaluation. Important details about selection statements are:

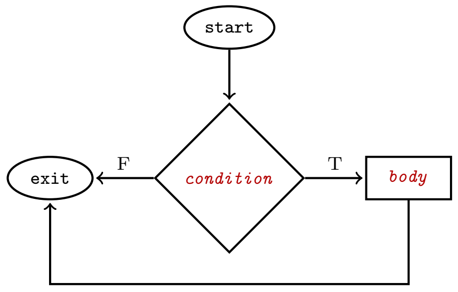
- A selection statement is a collection of special statements.
- Every statement of a selection statement has a body.
- The body of at most one statement of a selection statement can be executed.
- A selection statement executes at most once before returning to the rest of the code.
- Selection statements can be nested.

Simple If Statement

Whenever a set of statements needs to be executed only if a condition is met, a *simple if statement* is used. Its syntax is

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

Initially, the condition is evaluated. If the evaluation is true, the body is executed; otherwise, it is skipped. and the remaining code is executed.



Example:
The absolute value function

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

is illustrated in the code segment below

```
double x;
std::cin >> x;
if(x < 0) {x *= -1;}
std::cout << x << "\n";
```

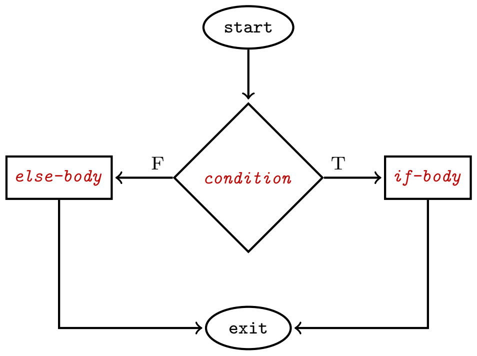
Although the mathematical version of the absolute value function has two conditions, only one output modifies the input; hence, a simple if statement will suffice.

If-Else Statement

Whenever one of two sets of statements needs to be executed based on a single condition evaluation, an *if-else statement* is required. In other words, use an if-else statement whenever there is an otherwise (either-or) situation. Its syntax is

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

It is a combination of an if statement followed by an else statement that **never** has a condition. Hence, if the condition is evaluated to be true, the if-statement body is executed; otherwise, the else-statement body is executed.



Example:
To determine if an integer is even or odd is an example of an otherwise situation; hence, the if-else statement is ideal as the code segment illustrates below

```
int n;
std::cin >> n;
std::cout << n << " is ";

if(n % 2 == 0) { std::cout << "even\n"; }
else { std::cout << "odd\n"; }
```

It is significant to mention to **never** use an if-else statement to display or assign a Boolean literal since it is needlessly redundant [it is echoing the condition evaluation].

Ternary Statements

For an otherwise situation that is a simple assignment, a *ternary statement* can be used instead of an if-else statement. Its syntax is

(*condition*)?*argument*:*argument*

such that the arguments **must be the same type**. Initially, the statement evaluates the condition. If the condition evaluation is true, the left argument is returned; otherwise, the right argument is returned.

Example: The previous example can be rewritten as

```
int n;
std::cin >> n;
std::cout << n << " is ";
std::cout << (n % 2 == 0)?("even\n"):("odd\n");
```

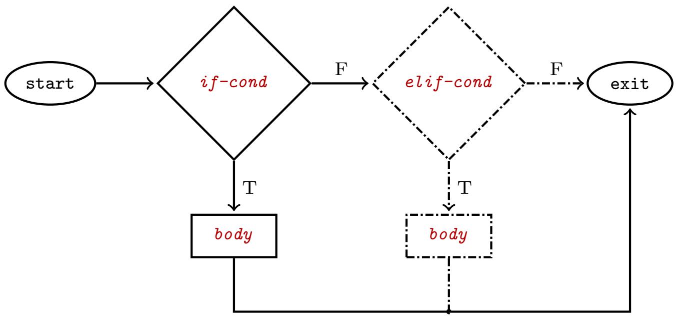
Although the ternary statement can be nested, avoid it since it is fallible and illegible.

Else-If Statements

Whenever one of several sets of statements needs to be executed based on a sequence of conditions such as in a multiple choice situation, an *else-if statement* is required. Its syntax is

$$\texttt{if(}\textit{condition}\texttt{) \{}\textit{body}\texttt{\}}\ \left[\texttt{else if(}\textit{condition}\texttt{) \{}\textit{body}\texttt{\}}\right]^{+}$$

It begins with an if statement followed immediately by at least one else-if statement, each containing a condition. It evaluates each condition in sequential order until it reaches a condition that evaluates to true, then, executes the body of the statement associated with that condition, or it terminates without executing any statement's body. No other condition is evaluated once a statement's body is executed.



where - - - - line means repeat 1 or more times.

Example:
To determine a letter grade is a multiple choice situation; hence, an else-if statement is ideal as the code segment illustrates below

Version 1

```
double grd;
char letter = 'F';
std::cin >> grd;

if(grd >= 90) { letter = 'A';}
else if(grd >= 80) { letter = 'B';}
else if(grd >= 70) { letter = 'C';}
std::cout << "Your grade is "<< letter << '\n';
```

Version 2

```
double grd;
char letter = 'F';
std::cin >> grd;

if(grd >= 90) { letter = 'A';}
else if(grd < 90 && grd >= 80) { letter = 'B';}
else if(grd < 80 && grd >= 70) { letter = 'C';}
std::cout << "Your grade is "<< letter << '\n';
```

It is significant to mention that the order of the statement's conditions matters if the conditions' truth set are not disjoint. For instance, in the previous example if `grd` were 89 both the second and third conditions of version 1's code will evaluate to true; however, the body of the second statement will be executed since it comes first in the sequence. The code would produce a different outcome if the conditions were arranged differently. However, rearranging the conditions in version 2's code, will not affect the outcomes because the conditions are mutually exclusive.

Ultimately, a selection statement is an if statement with optional parts. Hence, its full syntax is

$$\texttt{if(}\mathit{condition}\texttt{)}\ \{\mathit{body}\}\ \left[\texttt{else if(}\mathit{condition}\texttt{)}\ \{\mathit{body}\}\right]^{*}\ \left[\texttt{else}\ \{\mathit{body}\}\right]^{?}$$

It must start with an if statement and end with an else statement if included while else-if statements are placed in between.