

Recall Standard C++ supports a simple data type specialized for representing logical values.

`bool` type variables can have either of two values: `true` or `false`

The identifiers `true` and `false` are C++ reserved words.

In C++, in order to ask a question, a program makes an *assertion* which is evaluated to either `true` or `false` at run-time.

In order to assert "The student's age is above or equal to 21?", in C++:

```
const int LEGALAGE = 21;
bool isLegalAge;
int stuAge;
cin >> stuAge;
isLegalAge = (stuAge >= LEGALAGE );
```

The value of `isLegalAge` can now be tested to see if it is `true` or `false`.

Boolean expressions can, generally, take one of two forms.

The first is a relational expression, an expression (e.g., arithmetic) followed by a relational operator followed by another expression.

For example:  $( b * b - 4 * a * c ) > 0$

C++ has six standard relational operators:

The relational operators can be used to compare two values of any of the built-in types discussed so far.

Most mixed comparisons are also allowed, but frequently make no sense.

| Operator | Meaning                     |
|----------|-----------------------------|
| ==       | equals                      |
| !=       | does not equal              |
| >        | is greater than             |
| >=       | is greater than or equal to |
| <        | is less than                |
| <=       | is less than or equal to    |

Given:

```
const int MAXSCORE = 100;
char    MI = 'L', MI2 = 'g';
int     Quiz1  = 18, Quiz2  = 6;
int     Score1 = 76, Score2 = 87;
string  Name1  = "Fred", Name2 = "Frodo";
```

Evaluate:

```
Quiz1 == Quiz2
Score1 >= Score2
Score1 > MAXSCORE
Score1 + Quiz1 <= Score2 + Quiz2
```

```
MI == MI2
MI < MI2
'Z' < 'a'
```

```
Name1 < Name2
```

A logical expression consists of a Boolean expression followed by a Boolean operator followed by another Boolean expression (with negation being an exception).

C++ has three Boolean (or logical) operators:

| Operator | Meaning |
|----------|---------|
| !        | not     |
| &&       | and     |
|          | or      |

The Boolean operators && and || are binary, that is each takes two operands, whereas the Boolean operator ! is unary, taking one operand.

The semantics of the Boolean operators are defined by the following "truth tables":

| A     | !A    |
|-------|-------|
| true  | false |
| false | true  |

| A     | B     | A && B |
|-------|-------|--------|
| true  | true  | true   |
| true  | false | false  |
| false | true  | false  |
| false | false | false  |

| A     | B     | A    B |
|-------|-------|--------|
| true  | true  | true   |
| true  | false | true   |
| false | true  | true   |
| false | false | false  |

Given:

```
const int MINHEIGHT = 42, MAXHEIGHT = 54;
int FredsHeight, AnnsHeight;
int EmmasHeight = 45;
```

Evaluate:

```
MINHEIGHT <= EmmasHeight && EmmasHeight <= MAXHEIGHT
! ( EmmasHeight > MAXHEIGHT)
```

```
// When would the following be true? false?
FredsHeight < MINHEIGHT || FredsHeight > MAXHEIGHT
```

Two Boolean expressions are logically equivalent if they are both true under exactly the same conditions. Are the following two Boolean expressions logically equivalent?

```
!(EmmasHeight > FredsHeight)
```

```
EmmasHeight < FredsHeight
```

Suppose that A and B are logical expressions. Then DeMorgan's Laws state that:

$$\begin{aligned} \neg ( A \ \&\& \ B ) &\iff \neg A \ || \ \neg B \\ \neg ( A \ || \ B ) &\iff \neg A \ \&\& \ \neg B \end{aligned}$$

The Principle of Double Negation states that:

$$\neg ( \neg A ) \iff A$$

(The symbol  $\iff$  indicates logical equivalence.)

So the following negation:

```
!(FredHeight < MINHEIGHT || FredHeight > MAXHEIGHT)
```

...could be rewritten equivalently as:

```
(FredHeight >= MINHEIGHT && FredHeight <= MAXHEIGHT)
```

Since Boolean expressions can involve both arithmetic and Boolean operators, C++ defines a complete operator evaluation hierarchy:

0. Expressions grouped in parentheses are evaluated first.
1. (unary) `-` `!`
2. `*` `/` `%`
3. `+` `-`
4. `<=` `>=` `<` `>`
5. `==` `!=`
6. `&&`
7. `||`
8. `=`

Operators in groups (2) thru (7) are evaluated left to right, but operators in groups (1) and (8) are evaluated right to left.

Given:

```
int    i = 3, k = 5,  
       j = 0, m = -2;
```

Evaluate:

```
(0 < i) && (i < 5)  
(i > k) || (j < i)  
!(k > 0)
```

```
3*i - 4/k < 2  
i + j < k  
(i > 0) && (j < 7)  
(i < k) || (j < 7)  
(m > 5) || (j > 0)
```

Gotcha's:

```
k = 4           // confusing equality and assignment  
  
0 < i < 2       // allowed, but. . . it doesn't  
                // mean what you think. . .
```

Flow of Execution: the order in which the computer executes statements in a program.

Default flow is sequential execution:

```
cin >> x;  
y = x * x + x + 1;  
cout << y << endl;
```

Control structure: a statement that is used to alter the default sequential flow of control

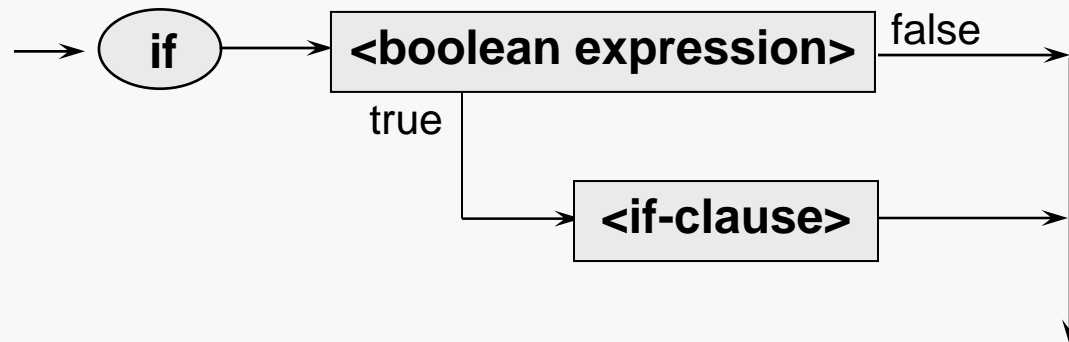
Selection: a control structure that allows a choice among two or more actions

The simplest selection structure in C++ is the `if` statement. Syntactically:



The Boolean expression must be enclosed in parentheses, and `<if-clause>` can be a single C++ statement or a compound statement.

The semantics of the `if` statement are:



The `if` statement is used to select between performing an action and not performing it:

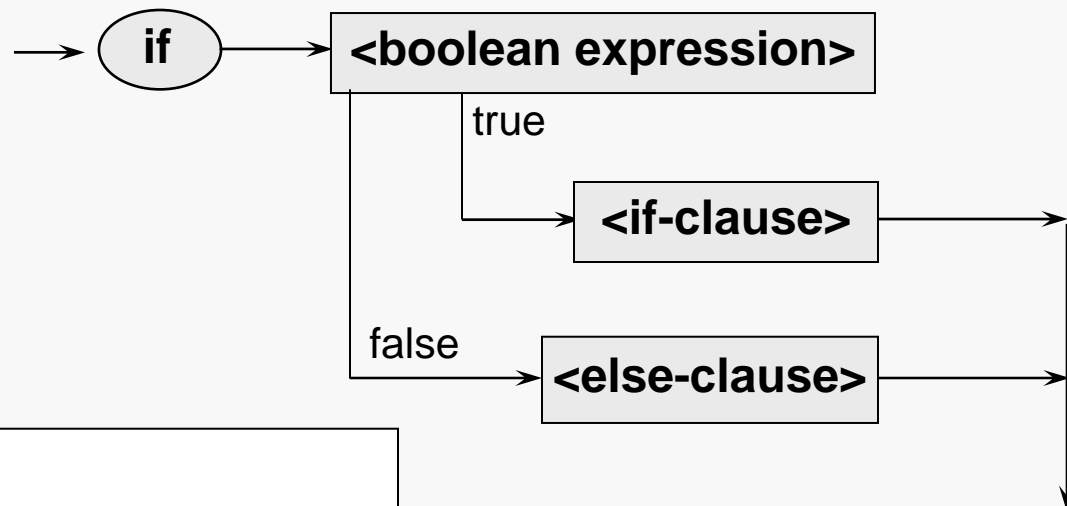
```
if (Grade == 'A') {  
    cout << "Good Job!";  
}
```

# Selection: if...else Statement

C++ also provides a selection structure for choosing between two alternatives, the if...else statement. Syntactically:



Semantically:



The if...else construct allows making an either-or choice:

```
if (Grade == 'A' ) {
    cout << "Good job!";
}
else {
    cout << "Grades aren't everything."
        << endl;
    cout << "But they help.";
}
```

The if-clause and else-clause may contain any valid C++ statements, including other if or if...else statements:


```
const double LOFLOOR = 100.0;
const double HIFLOOR = 500.0;
const double LORATE = 0.05;
const double HIRATE = 0.10;
double orderAmt;
. . .
if (orderAmt <= LOFLOOR) {
    Discount = 0.0;
}
else {
    if (orderAmt <= HIFLOOR) {
        Discount = LORATE * (orderAmt - LOFLOOR);
    }
    else {
        Discount = 20.0 +
            HIRATE * (orderAmt - HIFLOOR);
    }
}
```

**Conditions that are "mutually exclusive", (one condition being true excludes all others from being true), should be tested for with nested ifs, (as opposed to disjoint ifs), for efficiency.**

In some cases a problem may require a relatively large number of nested layers. In that case, the formatting used on the previous slide would cause the code to be poorly formatted. An alternative:

```
cout << "Your semester grade is ";

if (Average >= 90)
    cout << "A" << endl;
else if (Average >= 80)
    cout << "B" << endl;
else if (Average >= 70)
    cout << "C" << endl;
else if (Average >= 60)
    cout << "D" << endl;
else
    cout << "F" << endl;
```



**Note the layout and indenting style.**