Decision-Making in C

It is often necessary to perform one set of operations in one situation, and another set in a different situation.

C has statements that can perform decisions:

```
if
if-else
switch-case
```

The if statement

```
if(exp)s
```

Operation:

- The expression *exp* is evaluated. Usually, *exp* is a "logical expression"
- If the expression is true, then the statement *s* is executed.
- If the expression is false, *s* is ignored

Notes:

- 'if' is a keyword (lower case)

Logical Operators and Expressions

Logical operators produce the values "true" or "false"

```
<   - Less than
<=  - Less than or equal to
>   - Greater than
>=  - Greater than or equal to
==  - equal (NOT assignment!)
!=  - not equal
```

Examples:

```
a < b
x == 10.0
r+3 >= s+t
```

Some if Statement Examples

```
if(a > 10) a = 10;

bigger = a;
if(b > a) bigger = b;
cout << "The bigger value is " << bigger << endl;

if(x > y) x = y;

if(x > y)
   x = y;

if(x > y)
   x = y;
   y = 0;

if(x > y)
   {
    x = y;
    y = 0;
   }  // END if
```

Note: { } are used to create a "compound statement"

The if-else statement

`if(exp) s1 else s2`

The expression *exp* is evaluated. If the result is "true," then *s1* is executed. If the result is "false," then *s2* is executed.

Examples:

```
if(a > b)
   bigger = a;
else
   bigger = b;
cout << bigger;
```

```
if(x > y)
  {
    x = y;
    y = 0;
  }
else
  {
    y = x;
    x = 0;
  } // END if
```

```
if(x > y) {
    x = y;
    y = 0;
  }
else  {
    y = x;
    x = 0;
  } // END if
```

DECIS050

What Do "true" and "false" Mean?

Most of the time, logical expressions are used in if statements. However, any expression can be used:

If the expression value is 0, it is treated as "false"

If the expression value is non-zero, it is treated as "true."

When the computer performs a logical expression, it assigns a '1' for true.

```
if(a+10) b = c;
if (3) x = 1.0;
if (a = b) c = d;  // be careful with this one!
r = s >= t;
```

DECIS060

The bool Data Type

C++ implements the bool data type. It is specifically used for "true-false" values

```
bool y, n = false;
int a = 10, b = 5;

y = a < 10;
if(y)
   a = 0;
else
   b = 0;
```

The bool data type is named after George Boole, a 19th century mathematician who invented Boolean algebra, the theory behind all digital computers!

Connective Operators

The connective operators allow us to create more complicated logical expressions.

&&	("AND")
||	("OR")

```
if(a < b && c < d) x = 3;

if(x > 0.0 && x <= 10.0) y = x;

v = r || s && !t;
```

Operator Hierarchy (Update)

Highest	! + -	Unary NOT, unary plus, unary minus
	* / %	multiplication, division, modulus
	+ -	addition, subtraction
	< <= > >=	relational inequality operators
	== !=	relational equality operations
	&&	logical AND
	||	logical OR
Lowest	= += -= *= /= %=	assignment operators

Logical Expressions

```
in_range = x > -5.0 && x < +5.0;

out_of_range = x < -5.0 || x > +5.0;

out_of_range = !in_range;

isLetter = ((ch >= 'A') && (ch <= 'Z')) ||
           ((ch >= 'a') && (ch <= 'z'));

 even = n % 2 == 0;
```

Nested if Statements

```
if (fnlave >= 90.0)
   grade = 'A';
else
   if (fnlave >= 80.0)
      grade = 'B';
   else
      if (fnlave >= 70.0)
         grade = 'C';
      else
         if (fnlave >= 60.0)
            grade = 'D';
         else
            grade = 'F';
```

```
if (fnlave >= 90.0)
   grade = 'A';
else if (fnlave >= 80.0)
   grade = 'B';
else if (fnlave >= 70.0)
   grade = 'C";
else if (fnlave >= 60.0)
   grade = 'D';
else
   grade = 'F';
```

The switch-case Statement

Useful when multiple discrete options are possible

```
cout << "This is the " << num;
switch (num)
  {
   case 1:
           cout << "st option";
           break;
   case 2:
           cout << "nd option";
           break;
   case 3:
           cout << "rd option";
           break;
   case 4:
           cout << "th option";
  }  //END switch
cout << endl;
```

The switch-case Statement

default clause

```
cout << "This is the " << num;
switch (num)
  {
   case 1:
           cout << "st option";
           break;
   case 2:
           cout << "nd option";
           break;
   case 3:
           cout << "rd option";
           break;
   default:
           cout << "th option";
  }  //END switch
cout << endl;
```

The switch-case Statement

Multiple cases

```
cin >> ans;
switch (ans)
  {
   case 'n':
   case 'N':
            cout << "You answered no\n";
            break;
   case 'y':
   case 'Y':
            cout << "You answered yes\n";
            break;
   default:
            cout << "You didn't answer correctly\n";
  }  //END switch
```