

today: logical operations and control structures

- the if statement
- relational operators
- logical operators
- truth tables
- the while statement

the if branching statement

- we have already been using the if statement:

```
void moveNorth() {  
    // stop the roomba from trying to leave the room  
    if ( y < 11 ) {  
        y = y + 1;  
    }  
}
```

- let's look at this in a bit more detail...

the if branching statement

- the if is a *conditional*
 - means the computer makes a choice
- it is also a *control structure*
- general syntax:

```
if ( <something is true> ) {  
    <some instructions>  
}
```

Boolean expressions

- Boolean expressions are things that are either true or false
- Boolean variables: true (1) or false (0)
- Logical operators:

!	not
&&	and
	or

• example:

```
boolean x, y;
x = 1; // true
y = 0; // false
if ( x && y ) {
    cout << "this is false\n"; // this wouldn't happen...
}
if ( x || y ) {
    cout << "this is true\n";
}
```

truth tables

a	!a
false	true
true	false

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

relational operators

==	equality
!=	inequality
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

example:

```
int x, y;
x = -5;
y = 7;
```

some truths:

(x < y)	true
(x == y)	false
(x >= y)	false

the if branching statement (again)

```
// is the roomba still inside the room?
if (( x < 10 ) && ( y < 10 )) {
    cout << "the roomba is inside the room!\n";
}
```

the if-else branching statement

- a more efficient way of branching on more than one condition...
- instead of this:

```
if ( y < 10 ) {  
    y = y + 1;  
}  
if ( y == 10 ) {  
    y = 0;  
}
```

- do this:

```
if ( y < 10 ) {  
    y = y + 1;  
}  
else {  
    y = 0;  
}
```

- why is it more efficient?

general syntax for if-else

```
if ( <something is true> ) {  
    <some instructions>  
}  
else {  
    <alternative instructions>  
}
```

general syntax for if-else-if

```
if ( <something is true> ) {  
    <some instructions>  
}  
else if {  
    <first alternative instructions>  
}  
else if {  
    <second alternative instructions>  
}  
else {  
    <alternative instructions>  
}
```

the while looping statement.

- while allows us to repeat things:

```
while ( y <= 10 ) {  
    y = y + 1;  
}
```

general syntax of while

```
while ( <something is true> ) {  
    <some instructions>  
}
```

- this structure looks a lot like if
- how is it different?

summary

- we covered some of the basic control structures:
 - if, while
- along the way we looked at boolean expressions and relational operators as well.