

## INPUT, LOGICAL OPERATIONS, CONTROL STRUCTURES

### Today

- Input with `cin` statement
- The `if` statement
- Relational operators
- Logical operators
- Truth tables
- The `if-else` statement.
- The `while` statement

### Simple input

- We are already familiar with output:

```
int x;  
x = 5;  
cout << "The value of x is " << x << endl;
```

- C++ makes simple input just as straightforward:

```
int x;  
cout << "Enter a value for x: ";  
cin >> x;  
cout << "The value of x is " << x << endl;
```

(We'll look at more complex input later).

- The function that reads input is the `cin`.
- Note the use of the `>>`, which tells `cin` which variable to read the value into.
- The message:

```
cout << "Enter a value for x: ";
```

is called a *prompt*.

- A prompt tells the user what to do.

## Why we need control structures

- So far, all we have seen is how to make a program do a *sequence* of things.
  - goNorth()
  - goEast()
  - goNorth()
- There is more to life than this!
- We want to be able to make a program:
  - Choose between doing different things
  - Do the same thing several times
- C++ gives us *control structures* which allow us to do these things.

## The if branching statement

- Perhaps the simplest control structure is the `if` statement.
- Consider the robot from the homework.
- To tell when the robot is in the middle of the grid, we need to do:

```
if(x == 4) {  
    cout << "The robot is in the middle";  
}
```

- Let's look at it 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 structure:

```
if(<something that is true or false>  
{  
    <some instructions>  
}
```

## Boolean expressions

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

!	not
&&	and
	or

- Example:

```

boolean a, b;
x = 1; // true
y = 0; // false

if(x && y){
    cout << "This is false\n";
}

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 robot still in the world?

if ((x < 10) && (y < 10))
{
    cout << "The roomba is on the grid\n";
}

```

- And the actual code from the **roombaGame**:

```
if((x == dirtX)&&(y == dirtY))
{
    cout << "You found the dirt\n";
}
```

### The if-else branching statement

- A neater way of doing some branching.
- This:

```
if((x == dirtX)&&(y == dirtY)){
    cout << "You found the dirt\n";
}

if((x != dirtX)|| (y != dirtY)){
    cout << "You missed the dirt\n";
}
```

- Is a bit neater as:

```
if((x == dirtX)&&(y == dirtY))
{
    cout << "You found the dirt\n";
}
else
{
    cout << "You missed the dirt\n";
}
```

- General structure:

```
if(<something that is true or false>
{
    <some instructions>
}
else
{
    <alternative instructions>
}
```

## the while looping statement.

- while allows us to repeat things:

```
// Go north 4 times

count = 0;

while (count <= 4)
{
    goNorth();
}
```

- General structure:

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

- This structure looks a lot like if

## Summary

- We talked about simple input using `cin`.
- We covered some of the basic control structures:
  - if, while
- Along the way we looked at boolean expressions and relational operators as well.
- Now it is time to read Chapter 2 of the textbook.