

C++ fundamentals

Part 1

Gianluca Campanella

Variables and types

Variables

- Used to store mutable values
- Must have a **type** and a **name**

1

```
int x = 0;
```

Type	Use	Values
bool	Booleans	true, false
char	Characters	'a', 'b'
int	Integers	0, -1
double	Floating-point numbers	0.1, 1e-3

Arithmetic operators

Syntax	Operation
<code>a = b</code>	Assignment
<code>a + b</code>	Addition
<code>a - b</code>	Subtraction
<code>a * b</code>	Multiplication
<code>a / b</code>	Division
<code>a % b</code>	Modulo
<code>++a</code>	Prefix increment
<code>a++</code>	Postfix increment
<code>--a</code>	Prefix decrement
<code>a--</code>	Postfix decrement

Integers

`ints` store integers

```
1 int x = 2.5;
```

Watch for integer division!

```
1 int x = 2 / 3;  
2 double y = 2.0 / 3.0;
```

Comparison operators

Syntax	Operation
a == b	Equal to
a != b	Not equal to
a < b	Less than
a > b	Greater than
a <= b	Less than or equal to
a >= b	Greater than or equal to

Assignment and equality operators

Assignment operator

```
1 int x = 0;  
2 int y = x - 1;
```

Equality operator

```
1 int x = 0;  
2 int y = -1;  
3 bool z = y == x - 1;
```

Logical operators

Syntax	Operation
<code>!a</code>	$\neg a$
<code>a && b</code>	$a \wedge b$
<code>a b</code>	$a \vee b$
<code>a != b</code>	$a \oplus b$
<code>!a b</code>	$a \implies b$

Spot the difference!

What's the difference between

$a \neq b$ and $a = \neq b$?

Type conversion and casting

Type conversion (implicit)

```
1 double d = 3.5;  
2 int i = d; // Truncated to 3 without warning!  
3 cout << d << " " << i << endl;
```

Casting (explicit)

```
1 double d = 3.5;  
2 cout << d << " " << (int) d << endl;
```

Flow control

If statements

```
1  if (condition1)
2  {
3      // ...
4  }
5  else if (condition2)
6  {
7      // ...
8  }
9  else
10 {
11     // ...
12 }
```

Switch statements

```
1  switch (expression)
2  {
3      case constantValue1:
4          // ...
5      case constantValue2:
6          // ...
7          break;
8      default:
9          // ...
10 }
```

While loops

```
1 while (condition)
2 {
3     // ...
4 }
```

- **while** iterates until condition is **false**
- Careful with infinite loops!

For loops

```
1 for (initialisation; condition; update)
2 {
3     // ...
4 }
```

```
1 initialisation; // Executed only once
2 while (condition)
3 {
4     // ...
5     update;
6 }
```

Functions

Functions

- Functions encapsulate (reusable) bits of computation
- They're defined in terms of their arguments and return value

```
1 return_type function(type1 arg1, /*, ... */)
2 {
3     // ...
4     return value_of_return_type;
5 }
```

Functions

Example

```
1  int power(int x, int n); // Prototype
2
3  int power(int x, int n) // Declaration
4  {
5      int result = 1;
6      for (int i = 1; i <= n; i++)
7      {
8          result *= x;
9      }
10     return result;
11 }
```

Argument passing

Passing by value

- Argument values are **copied** by default
- Modifying them inside the function has no effect outside

Passing by reference

- Arguments can be referenced using **int&** x
- Modifying them inside the function changes them outside too!
- Use **const int&** x to prevent changes

Polymorphism

```
1 int power(int x, int n);  
2 double power(double x, int n);
```

- These two functions can coexist!
- The compiler 'knows' which one to call from the arguments:
 - power(2, 5) calls the first version
 - power(2.5, 2) calls the second version