COMP110: Principles of Computing

Transition to C++ I

Learning outcomes

By the end of this session you will

- Understand a thing
- Understand another thing
- Be convinced that LaTeX makes better-looking slides than PowerPoint

FALMOUTH
UNIVERSITY

Control structures

If statement

```
if (x > 0)
{
    std::cout << "x is positive" << std::endl;
}
else if (x < 0)
{
    std::cout << "x is negative" << std::endl;
}
else
{
    std::cout << "x is neither positive nor negative"
        << std::endl;
}
```

If statement

```
if (x > 0)
{
    std::cout << "x is positive" << std::endl;
}
else if (x < 0)
{
    std::cout << "x is negative" << std::endl;
}
else
{
    std::cout << "x is neither positive nor negative"
        << std::endl;
}
```

- Condition is always in parentheses ( )

Conditions

- Numerical comparison operators work just like Python:
  `==` `!=` `<` `>` `<=` `>=`
- Boolean logic operators look a little different

Python uses `and`, `or`, `not`

```
if not (x < 0 or x > 100) and not (y < 0 or y > 100):
    print "Point is in rectangle"
```

C++ uses `&&`, `||`, `!`

```
if (!(x < 0 || x > 100) && !(y < 0 || y > 100))
{
    std::cout << "Point is in rectangle" << std::endl;
}
```

Single-statement blocks

- In many cases, if a block contains only a single statement then the curly braces can be omitted

```
if (x > 0)
    std::cout << "x is positive" << std::endl;
else if (x < 0)
    std::cout << "x is negative" << std::endl;
else
    std::cout << "x is neither positive nor negative"
        << std::endl;
```

- Careless use of this can lead to difficult-to-find bugs

```
// This code is wrong!
if (z == 0)
    x = 0; y = 0;
```

Switch statement

```
switch (x)
{
case 0:
    std::cout << "zero" << std::endl;
    break;
case 1:
    std::cout << "one" << std::endl;
    break;
case 2:
    std::cout << "two" << std::endl;
    break;
default:
    std::cout << "something else" << std::endl;
    break;
}
```

While loop

```
while (x > 0)
{
    std::cout << x << std::endl;
    x--;
}
```

Do-while loop

```
do
{
    std::cout << x << std::endl;
    x--;
} while (x > 0);
```

Do-while loop

```
do
{
    std::cout << x << std::endl;
    x--;
} while (x > 0);
```

- `while` loop checks the condition **before** executing the loop body

Do-while loop

```
do
{
    std::cout << x << std::endl;
    x--;
} while (x > 0);
```

- `while` loop checks the condition **before** executing the loop body
- `do-while` loop checks the condition **after** executing the loop body

Do-while loop

```
do
{
    std::cout << x << std::endl;
    x--;
} while (x > 0);
```

- `while` loop checks the condition **before** executing the loop body
- `do-while` loop checks the condition **after** executing the loop body
- e.g. if `x == 0` to begin with, the `while` body does not execute, the `do-while` body executes once

For-each loop

```
std::vector<int> numbers { 1, 3, 5, 7, 9 };

for each (int x in numbers)
{
    std::cout << x << std::endl;
}
```

For-each loop

```
std::vector<int> numbers { 1, 3, 5, 7, 9 };

for each (int x in numbers)
{
    std::cout << x << std::endl;
}
```

- This works like the `for` loop in Python
- Used for iterating over data structures

For loop

```
for (int i=0; i<10; i++)
{
    std::cout << i << std::endl;
}
```

- In Python, this would be written as

```
for i in range(10):
    print i
```