CSCI251/CSCI851 Autumn-2020
Advanced Programming (**LT6**)

Lecture Tutorial 6

Goldilocks…

- There are various different approaches to this type of problem.
- It's important as an exercise to develop skills for use in A2.
- The idea is not simply to tell the story, or I could just capture everything in a literal string.
  - We want to be able to model the story and similar ones.
- Read the story.
  - Identify the characters and what they do.
  - What do the characters interact with?
    - Other characters? Items?

- Are there patterns, in particular repeated similar items or related behaviours?
  - Those likely indicate the need for classes.
- Bears: Three of them.
  - This suggests we have a `Bear` class to factor our the common ground between them.
  - They each have at least a name.
- Where does Goldilocks fit it?
  - She isn't a Bear, but she is a girl, so maybe we need a `Girl` class.
    - That may be a little limiting if we wanted to model similar scenarios so a `Human` class would give us more flexibility and allow us to use some of the same code for Hansel and Gretel too.

- We just put in a class that we will only need one instance of, should we go further?
- `Bear` and `Human` have name as a common attribute, and probably size (little, big, …).
- So we could set up a common base class both are derived from.
  - This will become possible through inheritance.

- Items: Chairs, porridge, bowl, beds, house, …
- We have multiple instances of most of these.
  - We might want classes to represent some of them.
  - We might want to generalise some (porridge→food?)
  - Note that chairs and beds are both particular types of furniture.
- Functionalities:
  - Bears and Humans can, in this story world anyway, both eat porridge and use chairs/beds …
  - They can all talk as well, more common ground.

- When do they eat/sleep/sit?
  - We might have characteristics representing hunger, tiredness …
  - We could have flags that record the state but having characteristics that are measured against might help.
- Note that we can have tests/interactions across classes/objects,
  - For example, to determine if Goldilocks eats some porridge we check:
    - If she is hungry enough to eat the volume of food.
    - How her food temperature tolerance compares against the temperature.
      - The food temperature isn’t fixed so waiting 10 minutes might allow that state, so there can be time dependence in the model.
      - What if Goldilocks had turned up when the Bears were at home?

From the lab:

- Syntax for throwing:

  ```
  throw(object);
  ```

  You are throwing an instance of a type, which may be an ADT or could be a primitive type.

- Private is a compiler setting.
- It doesn’t actually stop you from interacting with the location… ☹
- …

```
class thing {
public:
    int value1 = 5;
    void display(){cout << value2 << endl;}
private:
    int value2 = 77;
};

int main()
{
    thing A;
    cout << A.value1 << endl;
    cout << A.value2 << endl;
    cout << *(&A.value1 + 1) << endl;
    A.display();
    cin >> *(&(A.value1) + 1);
    A.display();
    return 0;
}
```

- We can do something similar to change `const`...

```
class thing {
public:
    int value1 = 5;
    const int value2 = 77;
};

int main()
{
    thing A;
    cout << A.value1 << endl;
    cout << A.value2 << endl;
    cin >> *(&(A.value1) + 1);
    cout << A.value2 << endl;
    return 0;
}
```

- What would happen if you swapped the order of the variables in thing?