Lab assignment 6: Standard Library containers

Dr Martin Porcheron

You have two weeks to complete this lab assignment—you must have it marked/signed off by a lab helper in a timetabled lab session. You must get this assignment marked by **11th March 2022** but I recommend you try and complete this as soon as possible. This assignment is worth 20 marks.

This lab task involves using C++ Standard Library containers.

It is not valid to use the line `using namespace std;` or any directive to that effect anywhere in your solution.

I recommend you keep all your lab assignments in a new directory. If working on the lab machines, create this within your home directory for CSC371. Keep each lab assignment in a separate directory.

Note: Do not copy and paste code from the PDF as it will contain non-standard characters and will not compile.

This lab assignment was originally written by Dr Joss Whittle.

Task 1: Implementing `std::hash` (8 marks)

In this exercise, you are going to need the `Cat` class from Lab Assignment 4. If you have not done so already for Lab 4:

1. In *cat.h* and *cat.cpp* implement a simple `Cat` class with a `std::string name` field and an `unsigned int lives` field.
2. Implement a constructor that takes in a name and lives value by `const` reference and uses them to initialise the `cat`.
3. Implement getters and setters for both name and lives, making sure to use `const` and references correctly to avoid implicit copying.
   - The specific setter behaviour described in Lab Assignment 4 is not needed for this lab, just the basic `Cat` class itself for holding its member variables and initialising them.
4. You do not need a destructor or copy/move constructor/assignment functions (`unsigned int` does not need them and `std::string` defines them for itself by default).

Now, for the specific task in this Lab Assignment:

1. Download this source code from the Canvas assignment page for lab assignment 6 or alternatively, create a file, *task1.cpp*, and type the following code into it:

   ```
   #include <iostream>
   #include "cat.h"

   int main(int argc, char* argv []) {
     Cat a("Garfield" , 42);

     std::cout << std::hash<Cat>()(a) << " = " << a << std:: endl;

     // Note the double ()() syntax!

     // std::hash <Cat>() default constructs an
     // object of type std::hash<Cat>

     // std::hash<Cat>()(a) invokes the call
     // operator () on that object with parameter

     // std::hash<Cat>()(a) returns an unsigned long integer

     return 0;
   }
   ```

   Do not modify this code.

2. In *cat.h* and *cat.cpp*: implement a friend `std::ostream` operator for writing a `Cat` with its name and lives to an output stream:

   ```
   std::ostream &operator<<(std::ostream &os, const Cat& cat)
   ```

3. In *cat.h* after the class definition, inject a `std::hash` operator into the `std` namespace. This will allow the `Cat` class to be used with associative C++ Standard Library containers.

   Copy the following code to get started. The code comment indicates where you will have to implement your hashing function.

   `hash<Cat>` is a functor and its call-function returns a `size_t`, which is an `unsigned int` type. Pay attention to this return type, and make sure you transform what you have to it as needed.

```
namespace std {
  template <>
  struct hash <Cat> {
    size_t operator()(const Cat &obj) const {
      // ... compute a hash as an unsigned
      // integer and return it...

      return some_computed_hash;
    }
  };
}
```

Hint: Use `std::stringstream` to make a unique string for a `Cat` based on its values and hand off hashing responsibility to `std::hash<std::string>()` which is already defined in the `<functional>` header.

You should use the fact you already have an `ostream` operator for the `Cat` class and call this!

This is not the most efficient hash function that can be made, but good hash function design is outside the scope of this course and could easily be an entire course to itself.

4. From the command line compile and link your program with the GCC compiler:

   ```
   $ g++ --std=c++11 cat.cpp task1.cpp -o task1
   ```

Expected output

If you run your program using the command:

```
$ ./task1
```

...you should get the output:

```
4296085865004304774 = Garfield has 42 lives...
```

Task 2: Implementing the equality operator (6 marks)

In this exercise you are going to modify your existing code to implement the `operator==` function.

1. Download this source code from the Canvas assignment page for lab assignment 6 or alternatively, create a file, *task2.cpp*, and type the following code into it:

```
#include <iostream>
#include <unordered_set>
#include "cat.h"

int main(int argc, char* argv []) {

  std::unordered_set <Cat> unique_cats;
  unique_cats.insert(Cat("Garfield",  42));
  unique_cats.insert(Cat("Garfield",  10));
  unique_cats.insert(Cat("Catbert",   666));
  unique_cats.insert(Cat("Toothless", 13));
  unique_cats.insert(Cat("Garfield",  42));
  unique_cats.insert(Cat("Garfield",  42));

  std::cout << std::endl << "Unique Cats" << std::endl;
  for (auto it = unique_cats.begin(); it != unique_cats.end(); it++) {
    std::cout << *it << std::endl;
  }

  return 0;
}
```

2. In *cat.h* and *cat.cpp* implement an equality `operator==`. Equality should test both name and lives are equal. This will have the prototype (you may want to make it a `friend`):

```
bool operator==(const Cat& lhs, const Cat& rhs);
```

3. From the command line compile and link your program with the GCC compiler:

```
$ g++ --std=c++11 cat.cpp task2.cpp -o task2
```

Expected output

If you run your program using the command:

```
$ ./task2
```

...you should get an output similar to (with maybe different ordering):

```
Unique Cats
Garfield has 42 lives...
Toothless has 13 lives...
Garfield has 10 lives...
Catbert has 666 lives...
```

Task 3: Implementing a comparison operator (6 marks)

In this exercise you are going to modify your existing code to implement the `operator<` function.

1. Download this source code from the Canvas assignment page for lab assignment 6 or alternatively, create a file, *task3.cpp*, and type the following code into it:

```
#include <iostream>
#include <set>
#include "cat.h"

int main(int argc, char* argv []) {

  std::set <Cat> unique_cats;
  unique_cats.insert(Cat("Garfield",  42));
  unique_cats.insert(Cat("Garfield",  10));
  unique_cats.insert(Cat("Catbert",   666));
  unique_cats.insert(Cat("Toothless", 13));
  unique_cats.insert(Cat("Garfield",  42));
  unique_cats.insert(Cat("Garfield",  42));

  std::cout << "Unique Ordered Cats" << std::endl;
  for (auto it = unique_cats.begin(); it != unique_cats.end(); it++) {
    std::cout << *it << std::endl;
  }

  return 0;
}
```

2. In *cat.h* and *cat.cpp* implement the less thsan comparison `operator<`. Less than should order based on name first, then lives if the names match. It has the prototype:

```
bool operator<(const Cat &lhs, const Cat &rhs)
```

3. Compile your program:

```
$ g++ --std=c++11 cat.cpp task3.cpp -o task3 -o task3
```

Expected output

If your run your program using the command:

```
$ ./task3
```

...you should get outputs the same as:

```
Unique Ordered Cats
Catbert has 666 lives...
Garfield has 10 lives...
Garfield has 42 lives...
Toothless has 13 lives...
```