Programming Studio 2 – C++

Week 8 Class 1

Dr Ruwan Tennakoon
Steven Korevaar

RMIT UNIVERSITY

Week 8 Class 1 Summary

Assignment Feedback

Inheritance and polymorphism

- Inheriting behaviour from other classes.

Operator Overloading

- Allowing you to use +, -, ==, etc with new classes.

Generics and Templates

- Allowing you to make classes/functions that accept unknown future types as parameters.

RMIT UNIVERSITY

Assignment Feedback

Video

- Students should have equal speaking times. Does not have to be live, prerecorded and edited is good.
- What data structures are you using AND WHY? Under what circumstances are you using the data structures, what makes your choice appropriate in terms of computation and memory efficiency?

Functionality

- **Menu**: should not crash no matter what gets entered
- **Maze CL input**: don't need to check if the maze is perfect, but you do need to check that there is only one exit, the format is correct (x and . only), the size matches the width and length entered.
- **Maze generation**: generates a perfect maze of the specified length and width (USING RECURSIVE BACKTRACKING) (reasonable locations only)
- **Solve manually**: should place the player randomly in the middle of the maze (is allowed access to maze information)
- **Show path to exit**: should show a path to the exit from where the player is currently standing (is not allowed any access to maze information, must read it all from the world). This must be implemented as right-hand wall-follower or BFS for the enhancement.

Tests

- Covers all major functionality of the assignment. Covers different many edge cases for user inputs. 2 files per test (input file, and expected output file)
- Format of tests are correct (name of files actually describes what is being tested)
- Each file should only be testing one edge case (don't make one pair of files for menu testing that tests all edge cases).
- Rough estimate of number of tests: 20

RMIT UNIVERSITY

Assignment Feedback

Code Style

Formatting:

- Code must be readable
- Proper indentation, and usage of white space
- No “magic numbers”, most numbers should be defined as #defines (except for things like 2, if checking for odd/even numbers).

Documentation:

- Comments for all lines of code that are not immediately self-evident as to their purpose
- Intuitive descriptions for large blocks of code (large blocks of code should also be placed in functions)

Abstraction:

- Logic for different functionality should be separated as much as possible (e.g., maze generation/solving logic should not be intertwined with menu logic)
- Utility files or classes should be used to store related functions and data (e.g., maze and agent classes)

Style Guide:

- Use of breaks (not inside switch-case statements) is forbidden.
- Functions should only have one return statement.

RMIT UNIVERSITY

Inheritance and polymorphism

Having classes inherit behaviour from other classes.

E.G., Animal class!

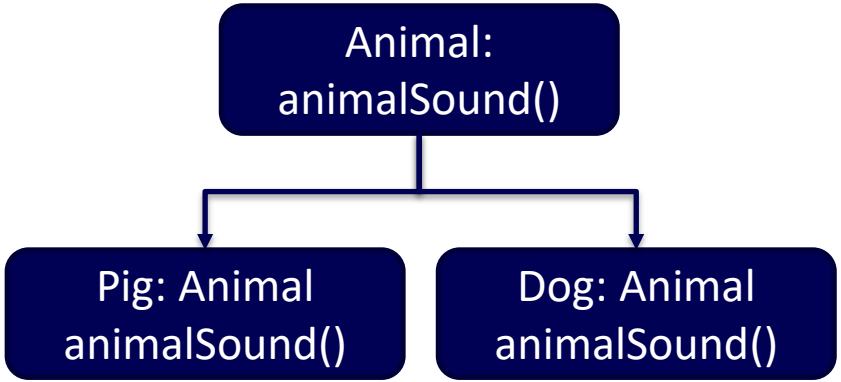
You can make a pig or a dog class inherit default behaviour from an animal class.

By default, any functions not reimplemented will be copied from a child's parent class.

i.e., the dog class' animalSound() function will (unless overwritten) call animal.animalSound().

When a method/function is overwritten by a child class, this is an instance of polymorphism

Useful when you have multiple but slightly different classes. Where you would have a lot of shared code.



RMIT UNIVERSITY

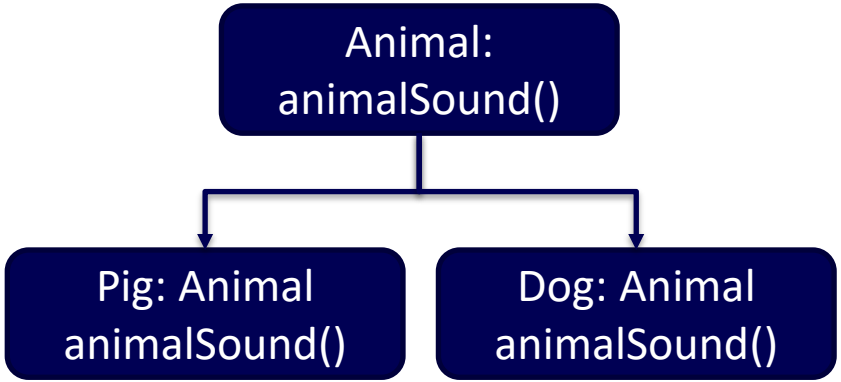
[https://www.w3schools.com/cpp/cpp_polymorphism.asp]

Inheritance and polymorphism

```
// Base class
class Animal {
  public:
    void animalSound() {
      cout << "The animal makes a sound \n";
    }
};

// Derived class
class Pig : public Animal {
  public:
    void animalSound() {
      cout << "The pig says: oink oink \n";
    }
};

// Derived class
class Dog : public Animal {
  public:
    void animalSound() {
      cout << "The dog says: bow wow \n";
    }
};
```



RMIT UNIVERSITY

[https://www.w3schools.com/cpp/cpp_polymorphism.asp]

Function and Operator Overloading

Function Overloading

When you have the same function can return differing types or accepts different parameters.

```
int test();
int test(int a);
float test(double a);
int test(int a, double b);
```

Operator Overloading

Where you redefine the default functionality of standard operators

- <<, [], +, -, ==: are the most common ones that are overloaded.

You've used them already in the assignment: coordinates can be added together with the + operator or compared with ==.

RMIT UNIVERSITY

Operator Overloading Example

Look into mcpp/src/util.cpp.

mcpp::Coordinate

```
Coordinate Coordinate::operator+(const Coordinate& obj)
const {
    Coordinate result;
    result.x = this->x + obj.x;
    result.y = this->y + obj.y;
    result.z = this->z + obj.z;
    return result;
}

bool Coordinate::operator==(const Coordinate& obj) const
{
    return (this->x == obj.x) && (this->y == obj.y) &&
(this->z == obj.z);
}
```

```
Coordinate Coordinate::operator-(const Coordinate&
obj) const {
    Coordinate result;
    result.x = this->x - obj.x;
    result.y = this->y - obj.y;
    result.z = this->z - obj.z;
    return result;
}

std::ostream& operator<< (std::ostream& out, const
Coordinate& coord) {
    out << "(" << coord.x << ", " << coord.y << ",
" << coord.z << ")";
    return out;
}
```

RMIT UNIVERSITY

Generics / Templates

What do we do if we want a function that computes the sum of two numbers?

```
int sum(int a, int b);
float sum(float a, int sum);
```

We may need to have a version for every combination of int/float/double/etc. Which is annoying and tedious.

We can use **“`template <typename T>`”** to have the compiler create new versions of functions as required with different data types.

```
template <typename TA, typename TB>
T pow(TA a, TB b){
    return a + b;
}
```

There are some limitations: you cannot guarantee that users of the template will provide types that are compatible with your functions. Ergo: contract programming.

RMIT UNIVERSITY

Generics / Templates

We can do the same thing with classes as well!

```
// Class template
template <class T>
class Number {
  private:
    // Variable of type T
    T num;
  public:
    // constructor
    Number(T n) : num(n) {}

    T getNum() {
        return num;
    }
};
```

```
// create object with int type
Number<int> numberInt(7);

// create object with double type
Number<double> numberDouble(7.7);
```

RMIT UNIVERSITY

https://www.programiz.com/cpp-programming/class-templates

Checkpoint 4 – Milestone 3

Final checkpoint!

This is the final period where we can help you with the assignment, ask as many questions as you can!

If you have team issues let us know asap.

Main components that should be mostly working:

- Random maze generation (recursive backtracking)
- Cleaning up when exiting the program

Aim to have the enhancements done as well so we can help you!

RMIT UNIVERSITY