Programming Studio 2 – C++

Week 7 Class 1

Dr Ruwan Tennakoon
Dr Steven Korevaar

RMIT UNIVERSITY

Week Class 1 Summary

Assignment Feedback

Smart Pointers

- Self-managed objects

Recursion

- Basics
- LinkedList Traversal Example

Week 7 Class 1 – Assignment Feedback

Black box testing

Tests should cover both normal use for functions and edge cases. These tests should define how you want your program to operate by the end of implementation at submission. They are not copy/pasted from your actual code (unless your code is 100% correct it will be obvious that you've done this).

Code Style

You will be marked on the legibility of your code and adhering to the style guide on Canvas. Something to keep in mind: please make sure your files are as minimal as possible, with functionality abstracted out. E.G., your main file should not handle all the logic for complex operations (reading mazes from CLI, generating random mazes, solving mazes, etc) these functions should be moved to separate files, so they can be easily read and understood in the context that is most appropriate to them. Having menu logic and maze functionality intertwined is very poor legibility.

This course is about writing programs that are correct and efficient (not just functional).

Need to consider the **data structures** used – need to be both memory and computational efficiency. For example, when remembering modifications to the world (to revert the changes) – what is the best data structure to hold the data? Is holding information about blocks that are never modified a useful use of resources?

Smart Pointers

Self-managed objects:

Objects that manage their own memory, meaning you no longer need to call delete, and worry about memory management when using these objects, only on implementing them.

Smart pointers

Memory-based self-managed objects, so you don't need to worry about memory leaks (as long as you don't do anything too weird).

Three versions:

1. Unique Pointers: Only one unique pointer can exist pointing to the same object (object ownership is important).
2. Shared Pointers: Many pointers can point to the same object (will only be deleted when all pointers are moved out of scope).
3. Weak Pointers: Basically, a raw pointer, you cannot guarantee the object it points to still exists. Allows you to break the rules! Use with caution (which is basically never).

Live coding: Using unique and shared pointers.

RMIT UNIVERSITY

Unique Pointers

Only one pointer may point to a unique pointer at a time!

This means you cannot assign a new pointer to point at an existing smart pointer.

Concepts of object ownership come into play here. If we want to move it around, we must use **`std::move().`**

Move transfers ownership of an object from one variable to another. HINT: use move when putting data into data structures.

```
std::unique_ptr<int> z = std::move(x);
```

You may also make references to a unique pointer in instances where you need to use the data, but this does not transfer ownership. Usually used for passing to functions.

```
std::unique_ptr<int> & z = x;

void printPtr(std::unique_ptr<int> & z);
```

As soon as the “owner” variable of a unique pointer moves out of scope or gets deleted, the underlying memory is deallocated.

RMIT UNIVERSITY

Shared Pointers

Many pointers can point to the same data.

Shared pointers keep a hidden value (“count”) that counts how many pointers point to the same object, once the count reaches zero (i.e., there are no more pointers to that object) the underlying data is deallocated.

```
std::shared_ptr<int> a = std::make_shared<int>(10);
std::shared_ptr<int> b = std::make_shared<int>(20);

std::cout << "Set a = b" << std::endl;
a = b;

a = nullptr;
b = nullptr;
```

RMIT UNIVERSITY

Recursion

Solving a problem by using the solution for a subset of the problem with a small additional step or value:

Factorial: n! = n*(n-1)!

Example: 5! = 5 * 4 * 3 * 2 * 1 = 15

Typically, we also define some "base case" which stops the recursion (i.e., 1! = 1)

```
int factorial(int num){
    int retValue = 1; //base case, 0! = 1
    if(num > 1) {
        //recursive or inductive step
        retValue = num * factorial(num-1);
    }

    return retValue;
}
```

RMIT UNIVERSITY

Week 7 Class 1 Exercises

Convert LinkedList to use smart pointers and recursion.

Consider using smart pointers for the assignment.

Continue working on the assignment.

Prepare for checkpoint 3: covers milestone 2

- generating the maze,
- cleaning up,
- solving the maze.

RMIT UNIVERSITY