Programming Studio 2 – C++

Week 6 Class 1

Dr Ruwan Tennakoon
Dr Steven Korevaar

RMIT UNIVERSITY

Week 6 Class 1 Summary

Download the starter code on canvas (Modules -> Week6 -> Class 1)

Object Ownership

- Field, Agent, and Game Controller classes

Debugging

- Basic concepts of memory (heap vs stack)
- Valgrind

Randomness

- Seeds, distributions

RMIT UNIVERSITY

Object Ownership

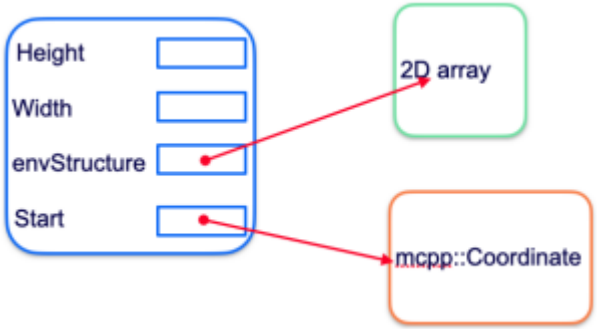
Core Concept: What parts of the code has responsibility for managing the memory that has been allocated to the heap.

Env Class from Starter Code:

What data is the Env class responsible for?

Where does this data come from?

What could happen if this data is mishandled?



Live Coding:

Destructor for the Env Class.

How do we delete 2D arrays?

Check success with Valgrind! `valgrind --tool=memcheck --leak-check=yes ./week01_example < env.input`

RMIT UNIVERSITY

Copying Pointers and Data

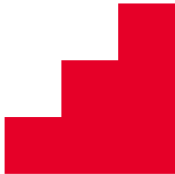
What is the difference between these two blocks of code?

```
int *apple = new int(10);
int *pear = new int(5);

pear = apple;

*apple = 301;

std::cout << "\t Apple: " << *apple << ", Pear: " << *pear << std::endl;
```

```
int *apple = new int(10);
int *pear = new int(5);

*pear = *apple;

*apple = 301;

std::cout << "\t Apple: " << *apple << ", Pear: " << *pear << std::endl;
```

What do they output and why?

RMIT UNIVERSITY



Copying Pointers and Data

Shallow copy:

- Copying surface level values.
- With pointers, this means copying the address not the data.

```
int *apple = new int(10);
int *pear = new int(5);
pear = apple;
```

Deep copy:

- Copying underlying data
- With pointers, this means dereferencing and copying the data, while making a new address.

```
int *apple = new int(10);
int *pear = new int(5);
*pear = *apple;
```

Live coding: basic shallow vs deep copying example with integers.

RMIT UNIVERSITY

Transferring Ownership

Sometimes we do want to copy pointers across:

If we have a StringManager object, we may want to create a string that it manages outside the class and pass it to the class. In this case the "ownership" of the string is transferred.

After a transfer of ownership, the new owner becomes responsible for deleting any allocated memory and the previous owner should no longer have access to those pointers.

```
#include <iostream>

class StringManager{
    public:
        char* str ;
        StringManager (char *) ;
        ~StringManager () ;
        void print() ;
};
```

```
StringManager::StringManager(char * str)  {
    this->str = str ;
    return ;
}

StringManager::~StringManager() {
    delete [] this->str ;
}

void StringManager::print() {
   std::cout << this->str << std::endl ;
}
```

```
int main() {
    char * str = new char[10] ;
    char c = 'a' ;
    for (int i = 0 ; i < 9 ; i++) {
        str[i] = c + i ;
    }
    str[9] = '\0' ;

    StringManager sm(str) ;
    sm.print() ;

    return 0;
}
```

RMIT UNIVERSITY

Debugging

Static Techniques:

- Code Style
- Static Code Analysis. Read source code and follow it the way the computer does (ie. mechanically, step-by-step) to see what it really does.
- Proving Code Correctness. Using static mathematical analysis

Dynamic Techniques:

- Test-then-Develop paradigm: Black-box testing and Unit Tests
- A program may be inspected “live" as it is executed. The C/C++ live-debugging tool is GDB. We can get VSCode to integrate with GDB. Requires CodeLLDB plugin. Also requires enabling debugging symbols when compiling (-g flag in compile command).
- Print statements!

RMIT UNIVERSITY

Debugging

Basic GDB:

1. Run “**gdb**” in WSL / Terminal.
2. Add output file: “**file <mazeRunner or whatever>**”.
3. Add breakpoints at a specific line of a file: “**break <.cpp file>:<line number>**”.
4. Add breakpoints at the start of a function: “**break <function name>**”.
5. Run the program: “**run**”.
6. Program will run until the first breakpoint.
7. Print out variables: “**print <variable name>**” (can dereference pointers with *).
8. Run the next line: “**step**” or “**next**” (step will move into functions, next will jump over functions).
9. Run until the next breakpoint: “**continue**”.

RMIT UNIVERSITY

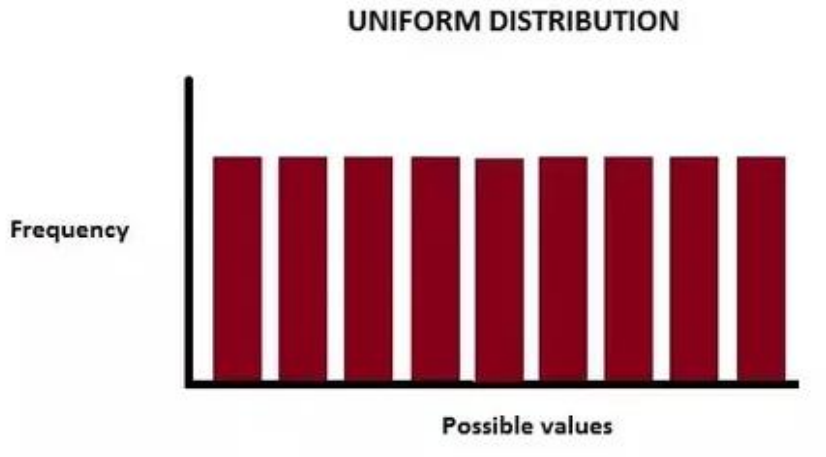
Randomness

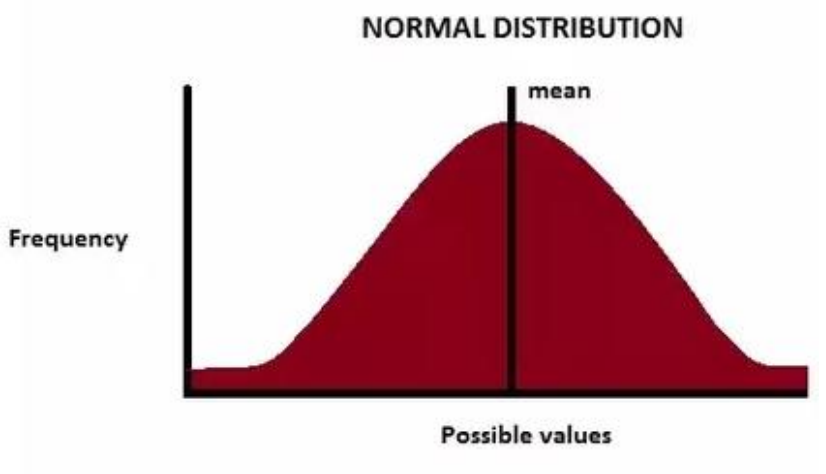
“True” randomness is where a sequence lacks any:

- Identifiable Pattern
- Predictability
- Ordering

C++ requires two entities (both defined in random header file).

- Engine - the algorithm that generates random numbers.
- Distribution - a mathematical formula that transforms the output of the engine into another sequence of value with varying properties





Week 6 Class 1 Exercises

Try and find all possible memory issues in the starter code!

Hints:

Whenever pointers are involved, double check edge cases!

Check for issues by trying to break things then running Valgrind.

```
valgrind --tool=memcheck --leak-check=yes ./week01_example < env.input
```

Make a copy constructor for the Env class

Hint: Remember shallow vs deep copying! Which should you be using?

Path Following:

Given the start location and a region of interest. Write a program to so that a player randomly walks around in the region of interest until he finds “treasure” (a gold block).

RMIT UNIVERSITY