Programming Studio 2
COSC2804

Week 05 - Class 01 Notes

Ruwan Tennakoon
ruwan.tennakoon@rmit.edu.au

Steven Korevaar
steven.korevaar@rmit.edu.au

Introduction

The aim of this class is to get set up to program Minecraft with C++, and revise some C++ concepts covered in Programming Boot-camp 2 via a Minecraft example. The key concepts covered include:

- Minecraft-C++ setup
- Basic C++ development cycle
- Reading and writing from the command line
- Declaration, definition, and initialization
- Arrays, Basic pointers, Classes
- Writing multi-file programs
- Automated compilation

Task:
During the class, we will implement a C++ program that *reads a 2D plan of a simple rectangular structure provided via the command line, and builds the structure in Minecraft world.*

The input will have the following structure:

```
<length width>
<structure>
<starting x-coordinate y-coordinate z-coordinate>
```

The structure is provided as a rectangular matrix of characters with 'x' representing a wall and '.' representing an empty space. An example input is given below.

```
5 6
xxxxxx
x....x
xxx.xx
x....x
xxxxxx
123 546 56
```

Consider the structure as the plan and you should build it to be two blocks tall using the block type "BRICK" in the location given by (starting x-coordinate y-coordinate z coordinate).

Tutorial

1. **Setting up C++ and Minecraft. Ask for help if you face any issues.** We expect every student to be ready with the programming environment by the end of the class. *Providing support for "setup" after this class would be challenging.*

   (a) If you haven't done so, please go through the instructions on this canvas page to install and setup Minecraft and C++.

   (b) To test the setup, let's write a simple program. Create a file called `week01_main.cpp` and write the following code in it to print "Hello World!".

   ```
   #include <iostream>

   int main(void){
       std::cout << "Hello World!" << std::endl;
       return EXIT_SUCCESS;
   }
   ```

   (c) Compile the program using the command line:

   ```
   g++ -Wall -Werror -std=c++17 -O -o week01_example week01_main.cpp [-lmcpp]
   ```

   - `Wall`: enable all error checking
   - `Werror`: convert warnings into errors, stopping compilation.
   - `-std=c++17`: enable all c++17 features
   - `-O`: turn on optimizer
   - `-o <filename>`: output filename of executable

   The C++ standard does define the behaviours, but it also allows for certain behaviours to be undefined, unspecified, or implementation-defined:

   - Undefined behaviour refers to behaviour that is not defined by the C++ standard. It can lead to unpredictable results and should be avoided. Examples of undefined behaviour include accessing an array beyond its bounds, division by zero, and dereferencing a null pointer.
   - Unspecified behaviour refers to behaviour that depends on the implementation of the C++ compiler. The C++ standard does not require the implementation to document its behaviour in these cases.
   - Implementation-defined behaviour refers to behaviour that is defined by the C++ standard but can vary between different implementations of the language. The implementation is required to document its chosen behaviour

   Therefore we will stick to the same compiler `g++`.

   (d) Run the program on the command line:

   ```
   ./week01_example
   ```

   (e) Next, let's test the Minecraft connection by placing a block at a predefined location in the Minecraft world.

   - Open Minecraft and select a coordinate on the ground to place a block. You can look for coordinates in Minecraft Java edition by pressing `F3` (or `Fn + F3` on Macs and some laptops or `Alt + Fn + F3` on newer Macs).
   - To place a block, we need our C++ program to communicate with Minecraft. For this, we need to establish a connection. We will use the `mcpp` library.

     ```
     mcpp::MinecraftConnection mc;
     ```

     Don't forget to include the mcpp header at the top of your code `#include <mcpp/mcpp.h>`

- The connection allows you to do things in the Minecraft world. Lets execute a terminal command in Minecraft:

```
mc.doCommand("time set day");
```

- MCPP also provides a variety of other useful commands. For example, it allows you to place a block in a given position.

```
void mcpp::MinecraftConnection::setBlock(const mcpp::Coordinate &loc,
                                         const mcpp::BlockType &blockType)
```

The function needs a `Coordinate` object and a `BlockType`. Let's create one and place a block:

```
mcpp::Coordinate startCoord(build_x, build_y, build_z);
mc.setBlock(startCoord, mcpp::Blocks::BRICKS);
```

- Final code in `week01_main.cpp`:

```
#include <iostream>
#include <mcpp/mcpp.h>

int main(void){
    std::cout << "Hello World!" << std::endl;

    // Create a Minecraft connection
    mcpp::MinecraftConnection mc;

    // set the time to day in Minecraft
    mc.doCommand("time set day");

    //Change [x], [y], [z] to the coordinate you selected previously
    int build_x = 144;
    int build_y = 71;
    int build_z = 148;
    mcpp::Coordinate coord( build_x, build_y, build_z);
    mc.setBlock(coord, mcpp::Blocks::BRICKS);

    return EXIT_SUCCESS;
}
```

- Compile the program as in the previous example and run.

2. **Reading (or writing) inputs (or outputs) via terminal**:

- Let's read the building coordinate via the terminal.

```
#include <iostream>
#include <mcpp/mcpp.h>

int main(void){
    // Create a Minecraft connection
    mcpp::MinecraftConnection mc;

    int build_x = 0;
    int build_y = 0;
    int build_z = 0;

    std::cout << "Enter the build coordinates [x y z]: " << std::endl;

    std::cin >> build_x;
```

```
        std::cin >> build_y;
        std::cin >> build_z;

        mcpp::Coordinate coord( build_x, build_y, build_z);
        mc.setBlock(coord, mcpp::Blocks::BRICKS);

        return EXIT_SUCCESS;
    }
```

  - For output (input), use the `cout` (`cin`) object. Contained in the `iostream` header (Within the `std` namespace)
  - output (input) operator:

```
<output location> << <what to output>
<input location> >> <variable>
```

  - `cin` Uses the type of the input variable to determine what to read from input.
  - `cin` is an object. Has functions to check for these things: `eof()` - check for end of file; `fail()` - check for read error.
- What if we don't want to keep typing inputs? Is there a way to get user inputs into a program? There is a terminal trick that can help - input redirection. We will be using this trick to test programs later.
  - Create a file (`env.input`) and write the inputs you want to feed into the program.
  - Redirect input

```
./week01_example < env.input
```

3. **Developing Modular Programs**. When collaborating on large and complex programs within a team setting, managing the codebase within a single file becomes impractical. It's essential to adopt a modular approach to programming—structuring the code across multiple files. This allows each team member to independently edit and maintain their respective components, enhancing overall efficiency and manageability.

   - Let's simulate a scenario involving multiple files. Although this particular example is quite simple and would typically be contained within a single file, it serves as a precursor to more complex cases we'll explore throughout the remainder of this course and its assignments.
   - In this example, we'll develop code to position blocks within a defined rectangular space in the Minecraft universe. The user will be prompted to provide details of the rectangle, including its length, width, and the coordinates of its upper-left vertex. Initially, we'll create a class to encapsulate these details and facilitate the construction of a rectangular structure in Minecraft.:

```
#include <iostream>
#include <mcpp/mcpp.h>

class Rectangle
{

public:
    Rectangle();
    ~Rectangle();

    void readRectangle(void);
    void buildRectangle(void);

private:
    int length; int width;
```

```
    int build_x; int build_y; int build_z;

};

Rectangle::Rectangle()
{
    length = 0; width = 0;
    build_x = 0; build_y = 0; build_z = 0;
}

Rectangle::~Rectangle()
{
}

void Rectangle::readRectangle(void){
    std::cout << "Enter the build coordinates [x y z]: " << std::endl;
    std::cin >> build_x;
    std::cin >> build_y;
    std::cin >> build_z;

    std::cout << "Enter the size of the rectangular Environment (L, W): "
                << std::endl;

    std::cin >> length;
    std::cin >> width;

}

void Rectangle::buildRectangle(void){
    mcpp::MinecraftConnection mc;
    for(int i=0; i<width; i++){
        for(int j=0; j<length; j++){
            mc.setBlock(mcpp::Coordinate(build_x+i,build_y,build_z+j)
                    , mcpp::Blocks::BRICKS);
        }
    }
}

int main(){
    Rectangle rect;
    rect.readRectangle();
    rect.buildRectangle();
    return EXIT_SUCCESS;
}
```

You can see that even this simple program results in a long code that is not convenient to manage. A more effective strategy for handling complex programs is to divide them into multiple files. Organizing the code into coherent segments and assigning each segment to a separate file enhances manageability. For instance, in the given example, we have two distinct segments: the main function and the Rectangle class. Let's separate them into multiple files.

- How to represent the Rectangle class? C++ has two types of files:
  - **Header Files** (.h/.hpp): The purpose of the header files is to describe to source files all of the information that is required in order to implement some part of the code. contain definitions, such as:
    - Function prototypes
    - Class descriptions
    - Typedef's
    - Common #define's

    Usually, Header files do not contain any code implementation
  - **Source Files** (.cpp): source code definitions/implementations.

- **C/C++ Preprocessor**: Prepare source code files for actual compilation

What happens if a header file is included multiple times? Preprocessor can help to identify such issues and prevent multiple includes.

- `#ifdef` - check if a definition exists
- `#ifndef` - check if a definition does not exist
- `#endif` - Close a #if check

- First least do the header. `Rectangle.h`

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include <iostream>
#include <mcpp/mcpp.h>

class Rectangle
{
public:
    Rectangle();
    ~Rectangle();

    void readRectangle(void);
    void buildRectangle(void);

private:
    int length; int width;
    int build_x; int build_y; int build_z;
};

#endif //RECTANGLE_H
```

- Next lets do the source file. `Rectangle.cpp`

```
#include "Rectangle.h"

Rectangle::Rectangle()
{
    length = 0; width = 0;
    build_x = 0; build_y = 0; build_z = 0;
}

Rectangle::~Rectangle()
{
}

void Rectangle::readRectangle(void){
    std::cout << "Enter the build coordinates [x y z]: " << std::endl;
    std::cin >> build_x;
    std::cin >> build_y;
    std::cin >> build_z;
    std::cout << "Enter the size of the rectangular Environment (L, W): "
                << std::endl;

    std::cin >> length;
    std::cin >> width;
}

void Rectangle::buildRectangle(void){
    mcpp::MinecraftConnection mc;
    for(int i=0; i<width; i++){
        for(int j=0; j<length; j++){
```

```
            mc.setBlock(mcpp::Coordinate(build_x+i,build_y,build_z+j)
                    , mcpp::Blocks::BRICKS);
        }
    }
}
```

- Finally we can link the new class to the main file.

```
#include <iostream>
#include <mcpp/mcpp.h>
#include "Rectangle.h"

int main(){
    Rectangle rect;
    rect.readRectangle();
    rect.buildRectangle();
    return EXIT_SUCCESS;
}
```

What happens at the third code line above?

- Now to compile multi-file Programs:

```
g++ -Wall -Werror -std=c++17 -O -o week01_example week01_main.cpp Env.cpp -lmcpp
```

- Do we have to type the command every time we compile? NO; we can use **Makefiles**. See lecture notes.
- Study the **Style Guide** on Canvas - there are few restrictions in this course - ignoring them will result in losing marks in the assignments.

Task Implementation

In this section, you will solve the task described on the first page.

1. First, Read the length and width of the environment to be built in Minecraft world.

```
int main(void){

    int envLength = 0;
    int envWidth = 0;

    std::cout << "Enter the size of the rectangular Environment (L, W): " << std::endl;

    std::cin >> envLength;
    std::cin >> envWidth;

    std::cout << "Length: " << envLength << ", Width: " << envWidth << std::endl;

    return EXIT_SUCCESS;
}
```

2. Next, read the structure of the building into a 2D primitive array.

```
char envStructure[envLength][envWidth];
char readChar;

for (int row = 0; row < envLength; row++) {
    for (int col = 0; col < envWidth; col++) {
        std::cin >> readChar;
```

```
            envStructure[row][col] = readChar;
        }
    }
```

How are 2D arrays stored in memory?

3. Print the structure, read above, to the terminal and see if the read is correct.

```
for (int row = 0; row < envLength; row++) {
    for (int col = 0; col < envWidth; col++) {
        std::cout << envStructure[row][col];
    }
    std::cout << std::endl;
}
```

4. Read building coordinate. This is the (x,y,z) of one of the corners of the rectangular building:

```
int build_x = 0;
int build_y = 0;
int build_z = 0;

std::cout << "Enter the building coordinates (X Y Z): " << std::endl;

std::cin >> build_x;
std::cin >> build_y;
std::cin >> build_z;
```

5. Build he structure in Minecraft:

```
mcpp::Coordinate startCoord(build_x, build_y, build_z);
for (int row = 0; row < envLength; row++) {
    for (int col = 0; col < envWidth; col++) {
        std::cout << envStructure[row][col];
        mc.setBlock(coord+mcpp::Coordinate(row, 0, col), mcpp::Blocks::AIR);
        if (envStructure[row][col] == 'x') {
            mc.setBlock(coord+mcpp::Coordinate(row, 0, col), mcpp::Blocks::BRICKS);
        }
    }
    std::cout << std::endl;
}
```

6. If you haven't already done so, move the reading of length and width into one separate function. How to return two values from a function?

   - Pointers:

```
void ReadEnvSize(int* envLength, int* envWidth){
    std::cout << "Enter the size of the rectangular environment (L, W): " << std::endl;
    std::cin >> *envLength;
    std::cin >> *envWidth;
}
```

   - References:

```
void ReadEnvSize(int& envLength, int& envWidth){
    std::cout << "Enter the size of the rectangular environment (L, W): " << std::endl;
    std::cin >> envLength;
    std::cin >> envWidth;
}
```

- Create an object (class) that holds both values and return that.

Discuss Declaration vs Definition vs Initialisation

- **Declaration**: Introduce a name (variable, class, function) into a scope; Fully specify all associate type information.
- **Definition**: Fully specify (or describe) the name/entity; All definitions are declarations, but not vice versa.
- **Initialisation**: Assign a value to a variable for the first time

7. Refactor reading the structure into a function. (Always make sure to pass in the array dimensions. Primitive arrays don't know the size).

```
void readEnvStdin(char** EnvStruct, int length, int width){
    char readChar;
    for (int row = 0; row < length; row++) {
        for (int col = 0; col < width; col++){
            std::cin >> readChar;
            envStructure[row][col] = readChar;
        }
    }
}
```

****Arrays are pointers**. To see this Lets change the main code to:

```
char **envStructure;
envStructure = new char*[envLength];
for(int i=0; i < envLength; i++){
    envStructure[i] = new char[envWidth];
}

readEnvStdin(envStructure, envLength, envWidth);
```

Memory issues: All memory created on Heap (`new` keyword) must be deleted. How can we check if there are uncleaned memory? **This is not good practice. You should not rely on debugging tools but identify issues when programming**

- Linux:

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

- Mac OS:

```
leaks --atExit -- ./week01_example < env.input
```

Need to clean memory allocated to the structure.

```
//delete memory
for(int i =0; i < envLength; i++){
    delete[] envStructure[i];
}
delete[] envStructure;
```

8. All the information we read so far is useful together. Therefore it is reasonable to combine them into a class - create an object with `size`, `building coordinate`, and `structure`.

   Let's create a class `Env` (keep simple by only including fields length and width for now).

   - Class Declaration:

```
class Env {
public:
    //constructors/de-constructors
    Env();
    Env(int length, int width);
    ~Env();

    //methods
    int getLength();
    int getWidth();

private:
    //Fields
    int length;
    int width;
};
```

- Class definition:

```
Env::Env(){
    this->length = 0;
    this->width = 0;
}

Env::Env(int length, int width) {
    this->length = length;
    this->width = width;
}

Env::~Env(){}

int Env::getLength() {
    return this->length;
}

int Env::getWidth() {
    return this->width;
}
```

- Create object:

```
// On stack
//Env testEnv();
Env testEnv(envLength, envWidth);

//On Heap
Env testEnv = new Env(envLength, envWidth);
...
delete testEnv;
```

Note: The class must be declared before the object is created - top of the file before main.

9. Move the Env class to a separate file.

```
#ifndef ENV
#define ENV

class Env {
public:
    //constructors/de-constructors
```

```
    Env();
    Env(int length, int width);
    ~Env();

    //methods
    int getLength();
    int getWidth();

private:
    //Fields
    int length;
    int width;
};

#endif //ENV
```

Process `#include` statements locates and includes header files

Now to compile multi-file Programs:

```
g++ -Wall -Werror -std=c++17 -O -o week01_example week01_main.cpp Env.cpp -lmcpp
```

10. Complete the task in page 1.

```
#include <mcpp/mcpp.h>
#include "Env.h"


void ReadEnvSize(int& envLength, int& envWidth);
void readEnvStdin(char** EnvStruct, int length, int width);
void ReadBuildLocation(int& build_x, int& build_y, int& build_z);

int main(void){
    std::cout << "Program started" << std::endl;
    mcpp::MinecraftConnection mc;
    mc.doCommand("time set day");

    int envLength = 0;
    int envWidth = 0;
    ReadEnvSize(envLength, envWidth);
    Env test_env(envLength, envWidth);
    std::cout << "Length: " << test_env.getLength() <<
                ", Width: " << test_env.getWidth() << std::endl;

    char readChar;
    for (int row = 0; row < envLength; row++) {
        for (int col = 0; col < envWidth; col++) {
            std::cin >> readChar;
            test_env.setEnvElement(row, col, readChar);
        }
    }

    int build_x = 0;
    int build_y = 0;
    int build_z = 0;
    ReadBuildLocation(build_x, build_y, build_z);
    std::cout << "Building at X: " << build_x << ", Y: " << build_y <<
                ", Z: " << build_z << std::endl;

    mcpp::Coordinate startCoord(build_x, build_y, build_z);
    for (int row = 0; row < envLength; row++) {
```

```
        for (int col = 0; col < envWidth; col++) {
            mc.setBlock(startCoord+mcpp::Coordinate(row, 0, col), mcpp::Blocks::AIR);
            if (test_env.getEnvElement(row, col) == 'x'){
                mc.setBlock(startCoord+mcpp::Coordinate(row, 0, col),
                            mcpp::Blocks::BRICKS);
            }
        }
    }

    return EXIT_SUCCESS;
}
```

```
#ifndef ENV
#define ENV

class Env {
public:
    //constructors/de-constructors
    Env();
    Env(int length, int width);
    ~Env();

    //methods
    int getLength();
    int getWidth();

    void setEnvElement(int, int, char);
    char getEnvElement(int, int);

private:
    //Fields
    int length;
    int width;
    char **envStructure;
};

#endif //ENV
```

```
Env::Env(){
    this->length = 0;
    this->width = 0;

    envStructure = new char*[length];
    for(int i =0; i < length; i++){
        envStructure[i] = new char[width];
    }
}

Env::Env(int length, int width) {
    this->length = length;
    this->width = width;

    envStructure = new char*[length];
    for(int i =0; i < length; i++){
        envStructure[i] = new char[width];
    }
}

Env::~Env(){
```

```
    for(int i=0; i < this->length; i++){
        delete[] this->envStructure[i];
    }
    delete[] this->envStructure;
}

int Env::getLength() {
    return this->length;
}

int Env::getWidth() {
    return this->width;
}

void Env::setEnvElement(int row, int col, char c) {
    this->envStructure[row][col] = c;
}

char getEnvElement(int row, int col){
    return this->envStructure[row][col];
}
```

Additional Exercises

1. Flatten the terrain and build your structure on it. You should only flatten an area equal to the area of your structure.
2. Update the code so that you do not need to input length and width. Infer this information from the given envStructure.