COMP710: Studio Session 03 – Exercise:

EXERCISE: C++ – *Using OO Inheritance*

Add a new C++ Project named "Using OO Inheritance" to your "SS03" Visual Studio Solution for this exercise.

Step 1: **Create the main…**

Create a **`main.cpp`** file, define a **`main`** function that prints **`"STARTING PROGRAM"`** to the console. Also ensure the **`main`** returns 0.

Step 2: **Declare the `Enemy` superclass…**

Add an **`enemy.h`** declaration as follows:

```
#pragma once
#ifndef __ENEMY_H__
#define __ENEMY_H__

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

    void MakeNoise() const;

protected:
    int m_x;
    int m_y;
};

#endif // __ENEMY_H__
```

Step 3: **Define the `Enemy` constructor…**

Add an **`enemy.cpp`** implementation file. Remember to **`#include "enemy.h"`** at the top of the **`enemy.cpp`** file. Add the definition for **`Enemy`** constructor, which initialises **`m_x`** and **`m_y`** both to 0, and then in the body of the constructor prints the following, followed by a newline:

```
"Constructor called on object at: " << this << " (Enemy)."
```

Note the **`this`** keyword evaluates to the be the address of the current object instance at runtime.

Step 4: **Define the Enemy destructor…**

Next add the definition for **`Enemy`** destructor, this simply prints the following, followed by a newline:

```
"Destructor called on object at: " << this << " (Enemy)."
```

Step 5: **Superclass `MakeNoise` method…**

Next, define the **`Enemy::MakeNoise`** method in **`enemy.cpp`**. This method must print the following message, where the current object's address is printed within the **`[]`** brackets, for example:

```
"Enemy [00000000] makes generic noise."
```

Step 6: **Avoid accidental `Enemy` copies…**

Add a **`private`** copy constructor declaration to **`enemy.h`** – this will stop any accidental copies of **`Enemy`** objects. Since this is **`private`**, there is no need to implement the copy constructor definition in the **`enemy.cpp`** file.

Step 7: **A basic container…**

In the **`main`** function, declare an array of **`Enemy*`** type, named **`basicContainer`** that has a size of **`5`** elements. Each element in this array will be used to store the address of an **`Enemy`** object instance. Also, use a **`constexpr int`** to store the size constant value of **`5`**.

Create five instances of **`Enemy`** objects using the **`new`** keyword – call **`new Enemy()`** five times, ensuring the pointer returned by each call to **`new`** is saved into the **`basicContainer`** elements 0 through 4.

Step 8: **Call a method…**

In the **`main`** function, call **`MakeNoise`** on each enemy instance – use a loop to iterate through the array of **`Enemy`** pointers, calling the method using the **`->`** operator.

Build and run your program, check the output, it should look as follows:

```
STARTING PROGRAM
Constructor called on object at: 00CCF9A8 (Enemy).
Constructor called on object at: 00CCF9F0 (Enemy).
Constructor called on object at: 00CCF7B0 (Enemy).
Constructor called on object at: 00CCFB58 (Enemy).
Constructor called on object at: 00CCFA38 (Enemy).
Enemy [00CCF9A8] makes generic noise.
Enemy [00CCF9F0] makes generic noise.
Enemy [00CCF7B0] makes generic noise.
Enemy [00CCFB58] makes generic noise.
Enemy [00CCFA38] makes generic noise.
```

Note the addresses where each object is stored will change each time you execute the program – hence they will certainly be different to the addresses shown in the sample above.

Step 9: **Checking the size…**

In the **`main`** function, before returning, print the **`sizeof(Enemy)`**.

Build and run your program, check the output matches the following:

```
STARTING PROGRAM
Constructor called on object at: 0085F8A8 (Enemy).
Constructor called on object at: 0085F598 (Enemy).
Constructor called on object at: 0085F9C0 (Enemy).
Constructor called on object at: 0085F7C8 (Enemy).
Constructor called on object at: 0085F608 (Enemy).
Enemy [0085F8A8] makes generic noise.
Enemy [0085F598] makes generic noise.
Enemy [0085F9C0] makes generic noise.
Enemy [0085F7C8] makes generic noise.
Enemy [0085F608] makes generic noise.
sizeof(Enemy) is: 8
```

Step 10: **Memory leaks…**

Before the end of the **`main`** function, avoid any memory leaks on exit by calling **`delete`** on the pointer to each object instance prior to returning.

Build and run your program, check the output now contains the printing from the destruction of the instances at the end of the program.

Step 11: **Inheritance – Creating a subclass…**

Declare a subclass of **`Enemy`**, named **`Zombie`** in **`zombie.h`**.

The **`zombie.h`** header file will need to **`#include "enemy.h"`** – note a forward declaration cannot be used here, as the **`Zombie`** subclass declaration must know the concrete size of the **`Enemy`** type that it is to inherit from. Ensure the **`Zombie`** inherits from the **`Enemy`**:

```
class Zombie : public Enemy
{
```

Step 12: **Zombie's constructor and destructor…**

Add the declaration and definition for **`Zombie`** constructor, that prints the following, followed by a newline:

```
"Constructor called on object at: " << this << " (Zombie)."
```

Next add the declaration and definition for **`Zombie`** destructor, this simply prints the following, followed by a newline:

```
"Destructor called on object at: " << this << " (Zombie)."
```

Step 13: **Override a member function (i.e.: method)…**

In the **`zombie.h`** class declaration, add a declaration for the **`MakeNoise`** method.

In the **`zombie.cpp`** file, implement **`Zombie::MakeNoise`** method, which prints the following message, where the object's address is printed within the **`[]`** brackets, for example:

```
"Zombie [00000000] makes OHHH, AHHHHHYAAAA noise."
```

Step 14: **Instantiate some zombies…**

In **`main.cpp`**, edit the **`Enemy*`** array declaration to increase the size – adding additional space for three more elements – do this by updating the **`constexpr int`**.

Next, instantiate three **`Zombie`** objects, saving each address return from **`new`** in elements 5 to 7.

Ensure the loop which iterates over the array and calls **`MakeNoise`** now also iterates over all eight pointers.

In the **`main`** function, before returning, print the **`sizeof(Zombie)`**.

Build and run your program, check the output matches the following:

```
STARTING PROGRAM
Constructor called on object at: 011EE3D0 (Enemy).
Constructor called on object at: 011EE5C8 (Enemy).
Constructor called on object at: 011EE1D8 (Enemy).
Constructor called on object at: 011EE248 (Enemy).
Constructor called on object at: 011EE328 (Enemy).
Constructor called on object at: 011EE088 (Enemy).
Constructor called on object at: 011EE088 (Zombie).
Constructor called on object at: 011EE590 (Enemy).
Constructor called on object at: 011EE590 (Zombie).
Constructor called on object at: 011EE600 (Enemy).
Constructor called on object at: 011EE600 (Zombie).
Enemy [011EE3D0] makes generic noise.
Enemy [011EE5C8] makes generic noise.
Enemy [011EE1D8] makes generic noise.
Enemy [011EE248] makes generic noise.
Enemy [011EE328] makes generic noise.
Enemy [011EE088] makes generic noise.
Enemy [011EE590] makes generic noise.
Enemy [011EE600] makes generic noise.
sizeof(Enemy) is: 8
sizeof(Zombie) is: 8
Destructor called on object at:  011EE3D0 (Enemy).
Destructor called on object at:  011EE5C8 (Enemy).
Destructor called on object at:  011EE1D8 (Enemy).
Destructor called on object at:  011EE248 (Enemy).
Destructor called on object at:  011EE328 (Enemy).
Destructor called on object at:  011EE088 (Enemy).
Destructor called on object at:  011EE590 (Enemy).
Destructor called on object at:  011EE600 (Enemy).
```

Notice that since the array is simply storing **`Enemy*`** calling **`MakeNoise`** calls only the superclass method, even if the some of the actual instances are of the subclass **`Zombie`** type!

Step 15: **Add polymorphic behaviour…**

Now add the **`virtual`** keyword to the **`enemy.h`** method declaration of **`MakeNoise`**.

When **`virtual`** is added to class declaration's method, it should also be added to the destructor. This will need to be added on the super and sub classes declarations to ensure object destruction occurs.

Build and run your program, check the output, it should look as follows:

```
STARTING PROGRAM
Constructor called on object at: 0111E520 (Enemy).
Constructor called on object at: 0111E440 (Enemy).
Constructor called on object at: 0111E478 (Enemy).
Constructor called on object at: 0111E638 (Enemy).
Constructor called on object at: 0111E4B0 (Enemy).
Constructor called on object at: 0111E558 (Enemy).
Constructor called on object at: 0111E558 (Zombie).
Constructor called on object at: 0111E590 (Enemy).
Constructor called on object at: 0111E590 (Zombie).
Constructor called on object at: 0111E6A8 (Enemy).
Constructor called on object at: 0111E6A8 (Zombie).
Enemy [0111E520] makes generic noise.
Enemy [0111E440] makes generic noise.
Enemy [0111E478] makes generic noise.
Enemy [0111E638] makes generic noise.
Enemy [0111E4B0] makes generic noise.
Zombie [0111E558] makes OHHH, AHHHHHYAAAA noise.
Zombie [0111E590] makes OHHH, AHHHHHYAAAA noise.
Zombie [0111E6A8] makes OHHH, AHHHHHYAAAA noise.
sizeof(Enemy) is: 12
sizeof(Zombie) is: 12
Destructor called on object at:  0111E520 (Enemy).
Destructor called on object at:  0111E440 (Enemy).
Destructor called on object at:  0111E478 (Enemy).
Destructor called on object at:  0111E638 (Enemy).
Destructor called on object at:  0111E4B0 (Enemy).
Destructor called on object at:  0111E558 (Zombie).
Destructor called on object at:  0111E558 (Enemy).
Destructor called on object at:  0111E590 (Zombie).
Destructor called on object at:  0111E590 (Enemy).
Destructor called on object at:  0111E6A8 (Zombie).
Destructor called on object at:  0111E6A8 (Enemy).
```

Notice the difference – specialised Zombie behaviour now occurs when **`MakeNoise`** is called, even though the base-class pointer type is used by the calls in the **`main`** function.

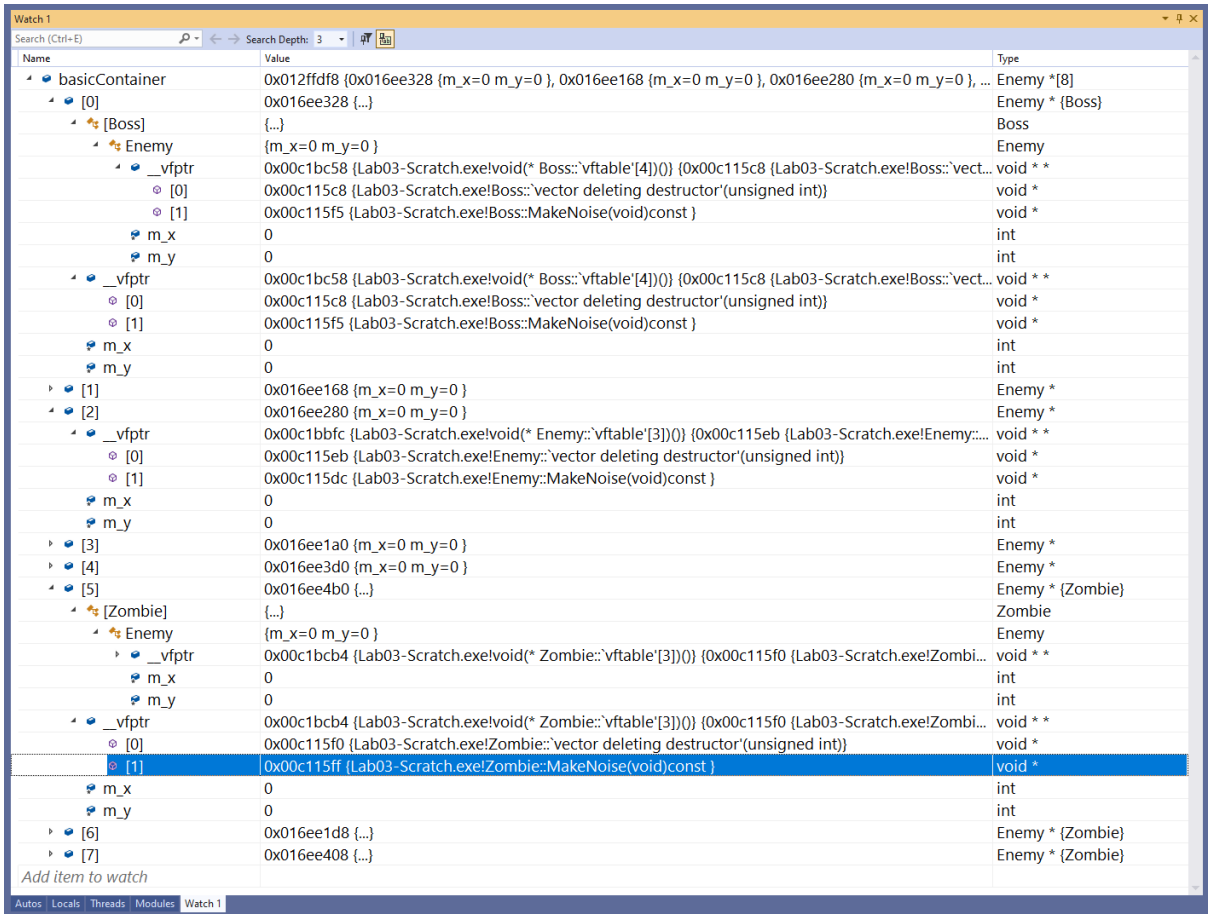
Also notice how the destruction of the **`Zombie`** through its base-class pointer now calls the **`Zombie`** destructor, then the **`Enemy`** destructor. This was not occurring on the previous step!

For source code robustness, in the **`zombie.h`** subclass declaration, add the **`virtual`** and **`override`** keywords to the **`MakeNoise`** method declaration. This will not change the program's behaviour, but it is good practice when using overridden **`virtual`** methods.

Also, now is a good opportunity to use the debugger and a breakpoint to pause program execution and to carefully step and check the **`basicContainer`** in the Watch Window.

Look for the V-Table member within the object's members and notice the different between the addresses of the **`Enemy`** instance method and the **`Zombie`** instance method.

For example:



You could also try removing the **`virtual`** keywords temporarily, build again and see what different this makes to the object's size in memory, as well as the Watch Window view.

Step 16: **Add another Enemy subclass, the Boss…**

Add a new subclass of **`Enemy`**, named **`Boss`**. Declare this in the **`boss.h`** file.

Define the **`boss.cpp`**, include a constructor, destructor, and **`MakeNoise`** method.

When constructed, the following is printed:

```
"Constructor called on object at: " << this << " (Boss)."
```

When destroyed, the following is printed:

```
"Destructor called on object at: " << this << " (Boss)."
```

When called, the **`Boss::MakeNoise`** method which prints the following message:

```
"Boss [00000000] makes GRRAAAAUUUU noise."
```

Finally, in **`main.cpp`**, replace one of the **`Enemy`** object instantiations with a **`Boss`** object and also, before returning, print the **`sizeof(Boss)`**.

Build and run your program, check the output, it should look as follows:

```
STARTING PROGRAM
Constructor called on object at: 0119E4B0 (Enemy).
Constructor called on object at: 0119E4B0 (Boss).
Constructor called on object at: 0119E558 (Enemy).
Constructor called on object at: 0119E018 (Enemy).
Constructor called on object at: 0119E0C0 (Enemy).
Constructor called on object at: 0119E248 (Enemy).
Constructor called on object at: 0119E210 (Enemy).
Constructor called on object at: 0119E210 (Zombie).
Constructor called on object at: 0119E3D0 (Enemy).
Constructor called on object at: 0119E3D0 (Zombie).
Constructor called on object at: 0119E0F8 (Enemy).
Constructor called on object at: 0119E0F8 (Zombie).
Boss [0119E4B0] makes GRRAAAUUUU noise.
Enemy [0119E558] makes generic noise.
Enemy [0119E018] makes generic noise.
Enemy [0119E0C0] makes generic noise.
Enemy [0119E248] makes generic noise.
Zombie [0119E210] makes OHHH, AHHHHHYAAAA noise.
Zombie [0119E3D0] makes OHHH, AHHHHHYAAAA noise.
Zombie [0119E0F8] makes OHHH, AHHHHHYAAAA noise.
sizeof(Enemy) is: 12
sizeof(Zombie) is: 12
sizeof(Boss) is: 12
Destructor called on object at:  0119E4B0 (Boss).
Destructor called on object at:  0119E4B0 (Enemy).
Destructor called on object at:  0119E558 (Enemy).
Destructor called on object at:  0119E018 (Enemy).
Destructor called on object at:  0119E0C0 (Enemy).
Destructor called on object at:  0119E248 (Enemy).
Destructor called on object at:  0119E210 (Zombie).
Destructor called on object at:  0119E210 (Enemy).
Destructor called on object at:  0119E3D0 (Zombie).
Destructor called on object at:  0119E3D0 (Enemy).
Destructor called on object at:  0119E0F8 (Zombie).
Destructor called on object at:  0119E0F8 (Enemy).
```

Step 17: **Initial random positions…**

Update the **`Enemy`** constructor, so that in the constructor's body sets the **`m_x`** member to a random value between **`0`** and **`120`**, inclusive; and sets the **`m_y`** member to a random value between **`0`** and **`30`**. Note the default size of the Console window in Windows is 120 characters wide, by 30 characters high – hence each enemy will start at a random location within the Console window.

Step 18: **Adding a generic Draw method…**

In the Enemy class, add a polymorphic **`Draw`** method to the **`enemy.h`** file using the **`virtual`** keyword, for example: **`virtual void Draw() const;`**

Next, in the **`enemy.cpp`**, file, implement the **`Enemy::Draw`** method such that it:

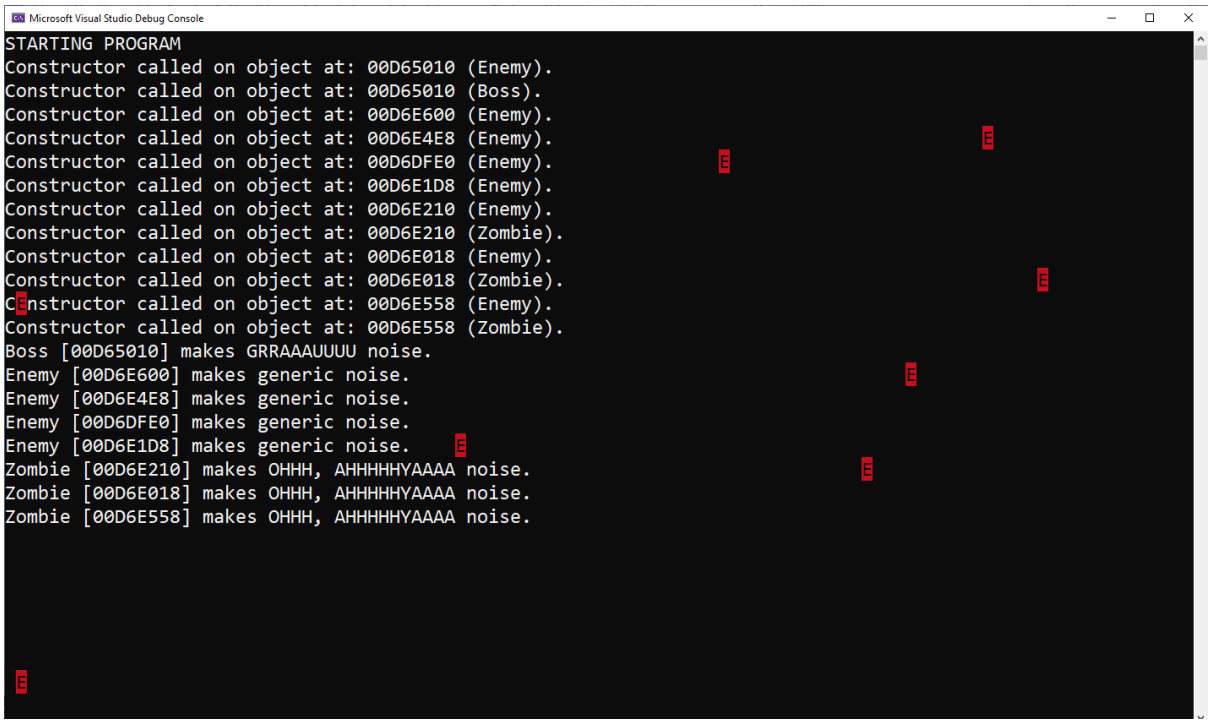
1. Moves the cursor to console coordinate (**`m_x`**, **`m_y`**)
2. Sets the text colour to be **`BLACK`** foreground text, with a **`RED`** background
3. Prints a single: **`"E"`**

Step 19: **Call the Draw method…**

Back in the **`main`** function, after the loop which calls the **`MakeNoise`** method on each instance, write another loop which again iterates through the array, but this time calls the **`Draw`** method of each object within the array.

You can also want to add a **`std::cin`** call to block and wait for user input after the calls to **`Draw`**.

Build and run your program. Check that your program outputs the eight enemies, each at random locations in the console window, for example as follows:



Step 20: **Override the `Draw` method…**

Declare and define an overridden polymorphic **`Draw`** method for each of the **`Enemy`** subclass types.

The **`Zombie`** must specialise by printing a **`"Z"`** with a **`CYAN`** background and **`BLACK`** text when its **`Draw`** method is called.

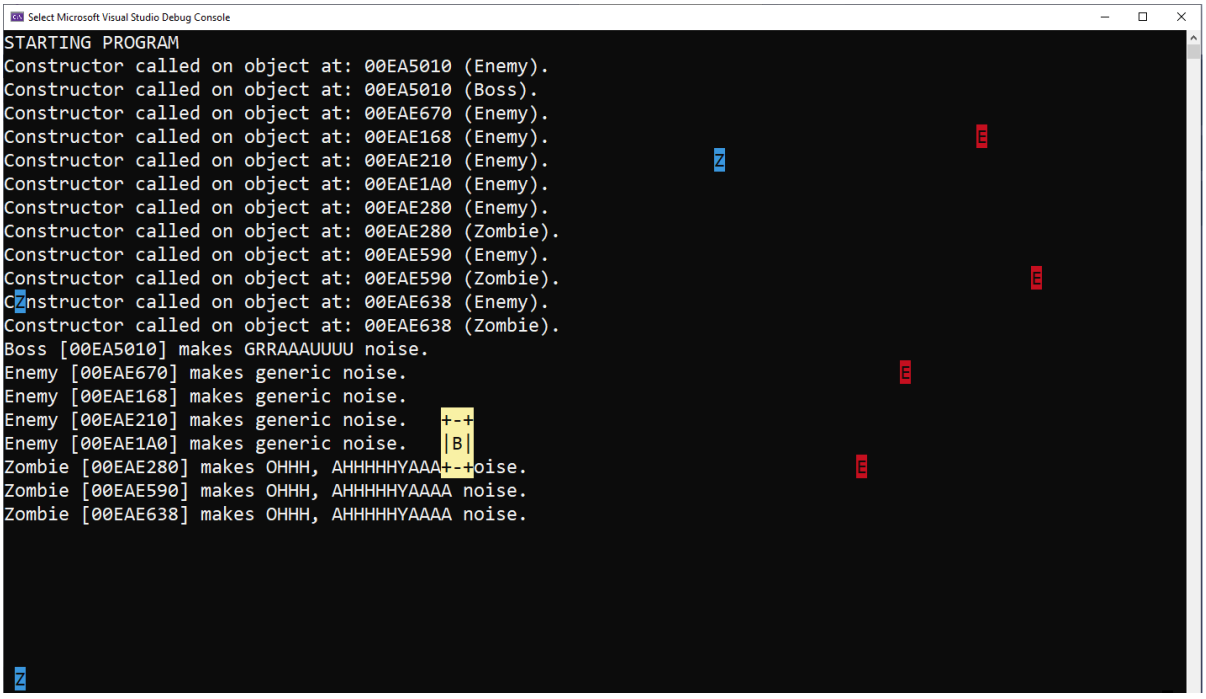
The **`Boss`** must specialise by printing a **`"B"`** in **`YELLOW`** background and **`BLACK`** text when its draw method is called, but also enclose the **`B`** with an ASCII art box in the following style:

```
+-+
|B|
+-+
```

You should use **`MoveCursorTo`** to help position the cursor prior to printing the text.

Remember to ensure the **`Draw`** method is declared as **`virtual`**, so that the subclass methodology is called when the base-class pointer is used to call the **`Draw`** method.

Build and run your program. Check that your program outputs the eight enemies, each at random locations in the console window, for example as follows:



Step 21: **Refactor the construction and destruction messages using a macro…**

Create a preprocessor macro that can be used to toggle printing of the text output by the constructors and destructors. This macro should controllable at compile time, allowing whether objects emit their construction and destruction message to the console by toggling the macro. This will be useful when debugging object lifespan, and also hiding the debug output when not needed.

Step 22: **Extend the program to create some basic game play…**

With the basic **`Enemy`** OO hierarchy created, you can now build out some more game features:

- Add a polymorphic **`Move`** method to the **`Enemy`** class, and its subclasses:
  - When called, a generic **`Enemy`** should move one position up/down or left/right from its current location – randomly choose the direction to move.
  - When called, a **`Zombie`** should choose to move horizontally or vertically one position, and then for the subsequent five calls to its specialised **`Move`** method, continue moving in the same direction. After five calls, the Zombie can then choose to randomly change direction. You may want to add a data member to the **`Zombie`** class that can count the number of moves made by the **`Zombie`**.
  - When called, a **`Boss`** should move instantaneously to a new randomly selected (x, y) location in the Console window.

- In the **`main`** function, add a simple "game loop" which calls **`Move`** for each enemy in the **`basicContainer`** array, followed by a call to **`Draw`** for each enemy in the array. This loop should also block, waiting for user input before the loop iterates again – and hence doesn't simply iterate over the calls to **`Move`** and **`Draw`** too quickly for the user to view the changes!

- Create a **`Player`** class, which can:
  - Be drawn to the Console window…
  - Move around the Console window world based upon user input…
  - Encounter and attack an enemy:
    - Create different encounter and attack behaviours for each type of enemy.
    - You should add relevant members to your classes to create this behaviour…
      - Including member data, and member functions (methods)…

- Create a game win / loss condition – and report the overall result to the user.

- Enhance the Console window output to create a "game world"…

- Enhance the **`MakeNoise`** output:
  - Position the text output by **`MakeNoise`** in a suitable location, rather than simply printing on the next line of Console output…
  - Refactor the **`MakeNoise`** calls from **`main`** to add these to a suitable place within the simple game loop…

- Consider where your **`Player`** class and **`Enemy`** class have any common characteristics that could be abstracted into a new superclass type which has "pure virtual" methods. If so, implement this design.

- Critically think about whether the **`basicContainer`** array could be changed to store concrete **`Enemy`** instances, rather than **`Enemy*`** values – what implications would this have on the ability to use polymorphic behaviour in this program? Could the game still work?

Once complete, commit your program's source code to your individual SVN folder – include the **`.sln`**, **`.vcxproj`**, **`.cpp`** and **`.h`** files, and ensure you do not commit any build output files.