Encapsulation & Inheritance & Polymorphism in C++

Lab 12

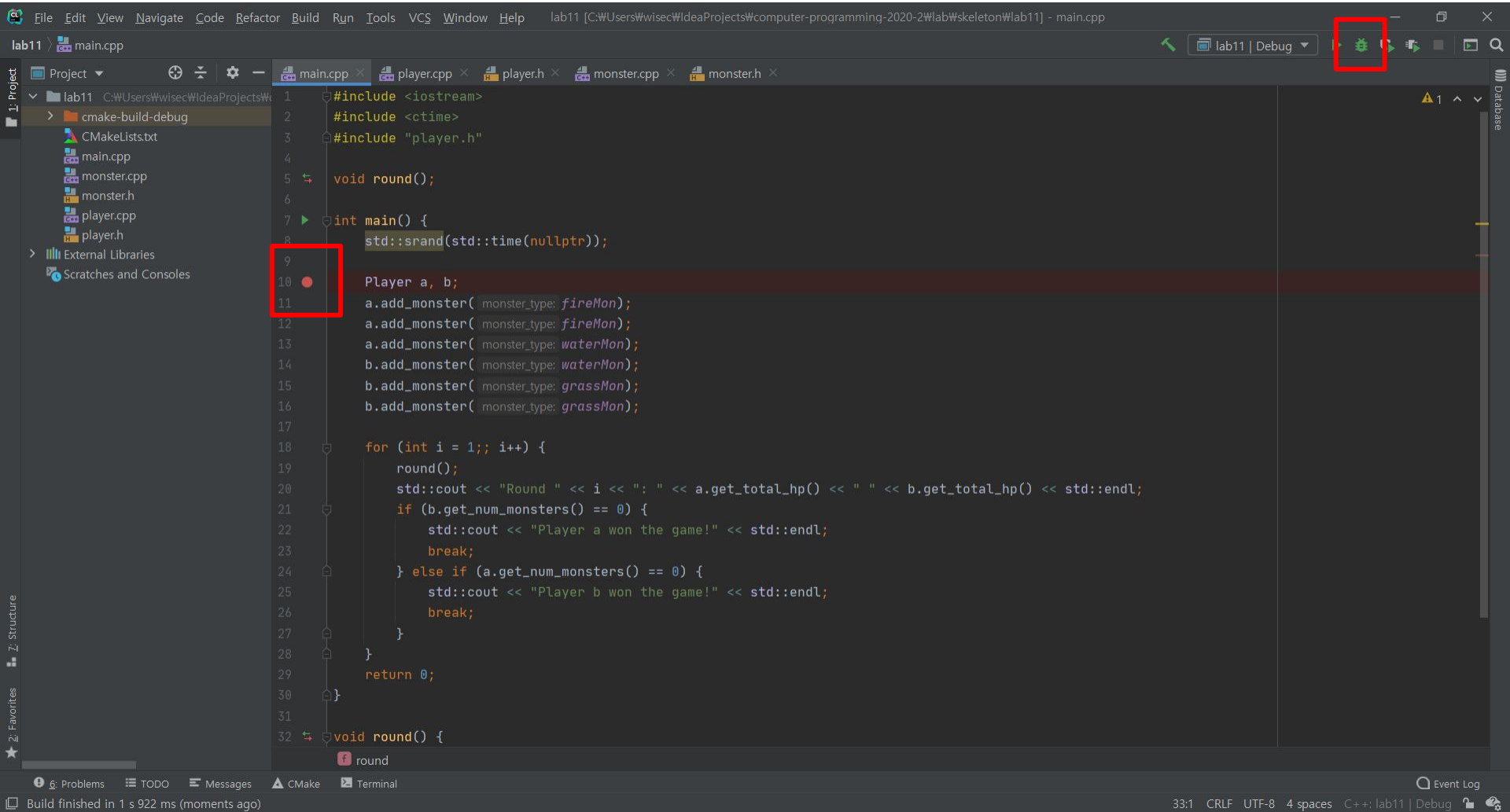
TA : Hyuna Seo, Kichang Yang, Minkyung Jeong, Jaeyong Kim

SEOUL NATIONAL UNIVERSITY

Announcement

- You should finish the lab practice and submit your job to eTL before the next lab class starts(Wednesday, 7:00 PM).
- The answer of the practice will be uploaded after the due.

Debugging in C++



Shortcut

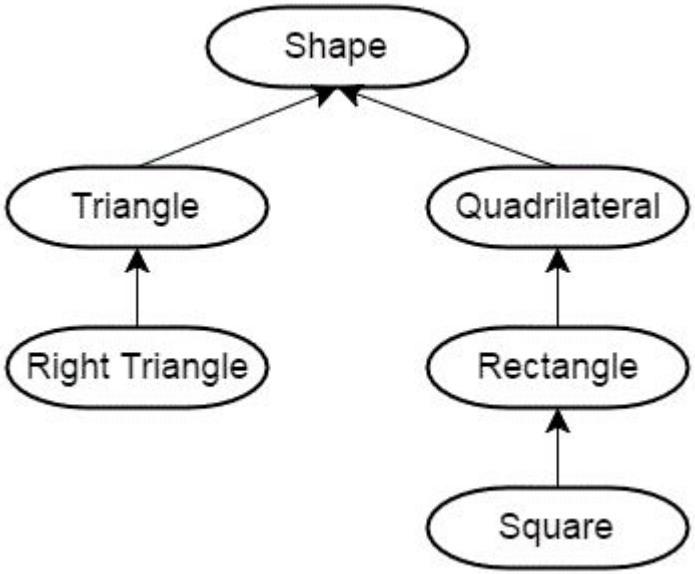
- Ctrl + F8 : Toggle Breakpoint
- Shift + F9 : Start Debugging
- F8 : Step Over
- F7 : Step Into
- Shift + F8 : Step Out
- F9 : Resume Program

Objectives

- Get used to C++ project structure. (*.cpp, *.h)
- Practice OOP with C++
  - ✓ Inheritance
    - ➢ Class extending
    - ➢ Method overriding
  - ✓ Encapsulation
    - ➢ Access modifiers
  - ✓ Polymorphism
    - ➢ Operator overriding

Recap: Inheritance

- Usually super(parent) class and derived(child) class has a is-a relationship
- Class derived-class: access-specifier base-class



Recap: Inheritance

```
class Shape {
 public:
  void setWidth(int w) {
     width = w;
  }
  void setHeight(int h) {
     height = h;
  }
 protected:
  int width;
  int height;
};

class Rectangle: public Shape {
 public:
  int getArea() { return (width * height); }
};
```

Recap: Inheritance

```
int main(void) {
    Rectangle Rect;

    Rect.setWidth(5);
    Rect.setHeight(7);

    //print the area of the object.
    Cout << “Total area: ” << Rect.getArea() << endl;

    Return 0;
};
```

Output

```
Total area: 35
```

Recap: Encapsulation

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

- A derived class inherits all base class methods with the following exceptions:
  - Constructors, destructors and copy constructors of the base class
  - Overloaded operators of the base class

Recap: Polymorphism

- Perform single action in many ways

```
class Animal {
 public:
  void animal_sound() {
     cout << “This animal makes this sound\n”;
  }
};

Class Cat: public Animal {
 public:
  void animal_sound() {
     cout << “Meow Meow\n”;
  }
};
```

Recap: Virtual Functions

- A virtual function is a member function in the base class that you redefine in the derived class.
- It tells the compiler to bind the function dynamically at runtime.
- This could be understood as C++'s way of implementing function overriding as in Java.
- It is declared using the `virtual` keyword.

Recap: Virtual Function Rules

- Cannot be static members.
- Are accessed through object pointers.
- Must be defined in the base class.
- Need to have an identical signature in both the base and the derived class.
  - same name and different signature → compiler considers them as overloaded functions

Example with Virtual Function

```
class A {
public:
    virtual void test() { cout << "A" << endl; }
};
class B: public A {
public:
    void test() { cout << "B" << endl; }
};
int main() {
    A a;
    a.test(); // A
    B b;
    b.test(); // B
    A* aptr = &b;
    aptr->test(); // B
}
```

Output
A
B
B

Pocketmon Game Application

- Some of you (Older students...) may have some memories of Pokemon games.
- We will make 2-player pocketmon game.

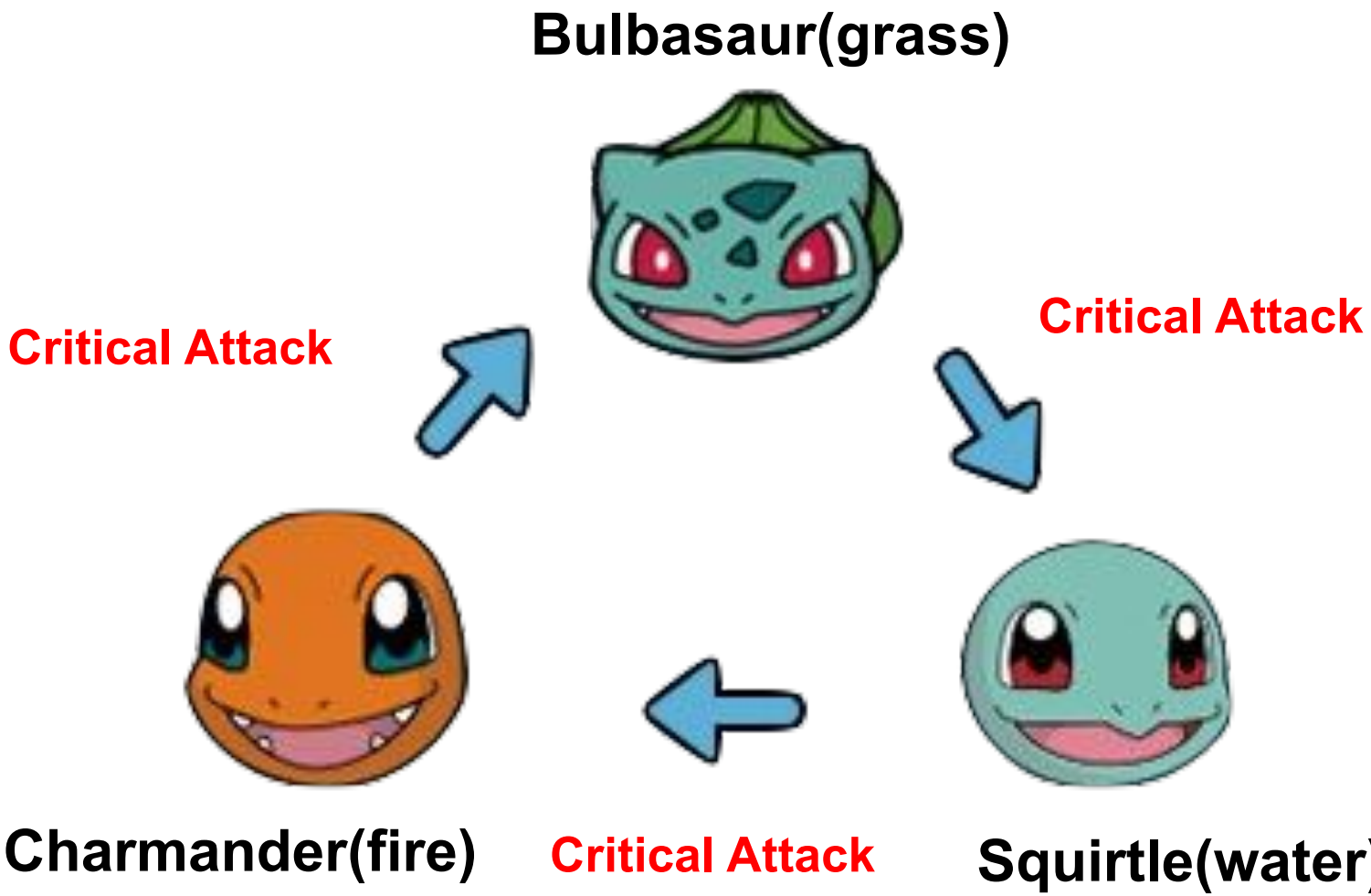
Pocketmon Game Application

- There are two players in the game.
- Each player can have at most 3 monsters.
- Monsters can be one of 3 types: fire, grass, and water
- At each round,
  a. Each player chooses one of the monster.
  b. Out of the 2 monsters, the monster that has higher speed stats will attack first and the other will attack back.
  c. If a monster's hp is below or equal to zero after a fight, the monster is excluded from the player.

Normal & Critical Attack

- The default attack damage is given for each type of monster.
- At each attack, the health of the target is decreased by the amount of the damage.
- Each type of monster has a counter monster type. WaterMon > FireMon > GrassMon > WaterMon ...
  - For example a WaterMon performs critical attack to a FireMon.
  - WaterMon performs normal attack to a GrassMon.
- Each type of monster has its own critical attack.

Normal & Critical Attack



Overview

- Problem 1: Implementing the base `Monster` class
  - Getting used to separating headers and implementations
- Problem 2: Implementing `critical_attack`
  - Practice function overriding by redefining `critical_attack` for every type of monster
- Problem3: Overriding “<<” operator for `Monster` class
  - Practice operator overriding by redefining << operator for all monsters
- Problem4: Implementing the `round()` function in main.cpp
  - Further practice on pointers, loops, and logic!

Problem 1

5 mins

- Add `id` attribute for `Monster` class objects by using `num_monsters` attribute in `Monster` class.
- Example
  - ✓ First created Monster object `id` : 0
  - ✓ Second : 1
  - ✓ Third : 2
  - ✓ …
- Add `speed` attribute and a getter method for the attribute for Monster class object so that we can decide who attacks first every round in the game.

Problem 2

10 mins

- Implement `critical_attack` method for `Monster`, `WaterMon`, `FireMon`, and `GrassMon`. `XXMon` should override `critical_attack` method of `Monster` class.
- `void critical_attack(Monster *attacked_monster)`
- `If the attacking monster is...`
  - Monster : 2 × damage
  - WaterMon : damage^2 / 2
  - FireMon : Random in range (0 ~ 10 × damage).
  - GrassMon : 3 × damage

You can use `std::rand()/RAND_MAX` in <cstdlib>.
`std::rand()` will return integer value between 0 ~ `RAND_MAX`.

Problem 3

5 mins

- Implement `<< operator` overriding for `Monster` so that the console prints out the Monster's name when called.
  - There is a name attribute given to the `Monster` class
  - Make sure you make the right changes to the header file too!

Problem 4

10 mins

- Implement `round()` function in main.cpp.
- You can change the signature of the function.
- At each round,
  - Each player chooses one of the monster.
  - Out of the 2 monsters, the monster that has higher speed stats will attack first and the other will attack back. (Assume no monsters have same speed)
  - If a monster's hp is below or equal to zero after receiving damage, the monster has fainted, thus excluded from the player.
  - If the monster is excluded from the player before getting a chance to attack, move onto next round.
  - Whenever a monster faints, please print whose pokemon fainted and the name of the pokemon that fainted.

Problem 4 (cont.)

10 mins

- You can use `Player::select_monster`, `Monster::attack`, and `Player::delete_monster` methods.

Submission

- Download skeleton files from eTL
- Compress your Project directory into a zip file.
- Rename your zip file as 20XX-XXXXX_{name}.zip
  - for example, 2020-12345_KimMinji.zip
- Upload it to eTL - Lab 11 assignment.