# 11.x — Chapter 11 comprehensive quiz

BY ALEX ON OCTOBER 29TH, 2016 | LAST MODIFIED BY ALEX ON JULY 9TH, 2018

# **Summary**

Inheritance allows us to model an is-a relationship between two objects. The object being inherited from is called the parent class, base class, or superclass. The object doing the inheriting is called the child class, derived class, or subclass.

When a derived class inherits from a base class, the derived class acquires all of the members of the base class.

When a derived class is constructed, the base portion of the class is constructed first, and then the derived portion is constructed. In more detail:

- 1. Memory for the derived class is set aside (enough for both the base and derived portions).
- 2. The appropriate derived class constructor is called.
- 3. The base class object is constructed first using the appropriate base class constructor. If no base class constructor is specified, the default constructor will be used.
- 4. The initialization list of the derived class initializes members of the derived class.
- 5. The body of the derived class constructor executes.
- 6. Control is returned to the caller.

Destruction happens in the opposite order, from most-derived to most-base class.

C++ has 3 access specifiers: public, private, and protected. The protected access specifier allows the class the member belongs to, friends, and derived classes to access the protected member, but not the public.

Classes can inherit from another class publicly, privately, or protectedly. Classes almost always inherit publically.

Here's a table of all of the access specifier and inheritance types combinations:

Access specifier in base class	Access specifier when inherited publicly	Access specifier when inherited privately	Access specifier when inherited protectedly
Public	Public	Private	Protected
Private	Inaccessible	Inaccessible	Inaccessible
Protected	Protected	Private	Protected

Derived classes can add new functions, change the way functions that exist in the base class work in the derived class, change an inherited member's access level, or hide functionality.

Multiple inheritance enables a derived class to inherit members from more than one parent. You should avoid multiple inheritance as much as possible.

# **Quiz Time**

1) For each of the following programs, determine what they output, or if they would not compile, indicate why. This exercise is meant to be done by inspection, so do not compile these (otherwise the answers are trivial).

1a)

```
1
     #include <iostream>
2
3
     class Base
4
     {
5
     public:
6
          Base()
7
8
              std::cout << "Base()\n";</pre>
9
          ~Base()
11
          {
```

```
12
               std::cout << "~Base()\n";</pre>
13
14
      };
15
16
      class Derived: public Base
17
      {
18
      public:
19
           Derived()
20
 21
               std::cout << "Derived()\n";</pre>
22
           }
23
      ~Derived()
24
 25
              std::cout << "~Derived()\n";</pre>
 26
           }
27
      };
28
29
      int main()
 30
      {
31
           Derived d;
32
      }
Hide Solution
```

Construction happens in order from most-Parent to most-Child. Destruction happens in the opposite order.

```
Base()
Derived()
~Derived()
~Base()
1b)
 1
      #include <iostream>
 2
 3
      class Base
 4
      {
      public:
 5
 6
          Base()
 7
 8
              std::cout << "Base()\n";</pre>
 9
10
      ~Base()
11
12
             std::cout << "~Base()\n";</pre>
          }
13
14
      };
15
16
      class Derived: public Base
17
      {
18
      public:
19
          Derived()
20
21
               std::cout << "Derived()\n";</pre>
22
          }
23
          ~Derived()
24
25
               std::cout << "~Derived()\n";</pre>
26
          }
27
      };
28
29
      int main()
30
31
          Derived d;
32
          Base b;
33
      }
```

Hint: Local variables are destroyed in the opposite order of definition.

#### **Hide Solution**

```
First we construct d, which prints:
```

```
Base()
Derived()
```

Then we construct b, which prints:

```
Base()
```

Then we destruct b, which prints:

```
~Base()
```

Then we destruct d, which prints:

```
~Derived()
```

~Base()

```
1c)
1
      #include <iostream>
 2
 3
      class Base
 4
 5
      private:
          int m_x;
 6
 7
      public:
 8
           Base(int x): m_x(x)
 9
               std::cout << "Base()\n";</pre>
10
11
          }
          ~Base()
12
13
          {
14
               std::cout << "~Base()\n";</pre>
15
16
17
          void print() { std::cout << "Base: " << m_x << '\n'; }</pre>
18
      };
19
20
      class Derived: public Base
21
      {
22
      public:
23
          Derived(int y): Base(y)
24
25
               std::cout << "Derived()\n";</pre>
26
          }
27
          ~Derived()
28
29
               std::cout << "~Derived()\n";</pre>
          }
30
31
32
          void print() { std::cout << "Derived: " << m_x << '\n'; }</pre>
33
      };
34
35
      int main()
36
37
           Derived d(5);
38
           d.print();
39
      }
```

### **Hide Solution**

Doesn't compile, Derived::print() can't access private member m\_x

```
1d)
```

```
1
     #include <iostream>
2
3
     class Base
4
     {
5
     protected:
6
         int m_x;
7
     public:
8
         Base(int x): m_x(x)
9
10
          std::cout << "Base()\n";</pre>
11
          }
12
         ~Base()
13
14
             std::cout << "~Base()\n";</pre>
15
16
17
         void print() { std::cout << "Base: " << m_x << '\n'; }</pre>
18
     };
19
20
     class Derived: public Base
21
22
     public:
23
         Derived(int y): Base(y)
24
25
              std::cout << "Derived()\n";</pre>
26
27
         ~Derived()
28
29
              std::cout << "~Derived()\n";</pre>
30
31
32
         void print() { std::cout << "Derived: " << m_x << '\n'; }</pre>
33
     };
34
35
     int main()
36
37
         Derived d(5);
38
         d.print();
39
     }
```

# **Hide Solution**

```
Base()
Derived()
Derived: 5
~Derived()
~Base()
```

# 1e)

```
1
     #include <iostream>
2
3
     class Base
4
     {
5
     protected:
6
          int m_x;
7
     public:
8
          Base(int x): m_x(x)
9
              std::cout << "Base()\n";</pre>
10
```

```
11
          }
12
          ~Base()
13
              std::cout << "~Base()\n";</pre>
14
15
16
          void print() { std::cout << "Base: " << m_x << '\n'; }</pre>
17
18
     };
19
20
     class Derived: public Base
21
     {
22
     public:
23
          Derived(int y): Base(y)
24
25
              std::cout << "Derived()\n";</pre>
26
          }
27
          ~Derived()
28
29
              std::cout << "~Derived()\n";</pre>
30
31
32
          void print() { std::cout << "Derived: " << m_x << '\n'; }</pre>
33
     };
34
35
     class D2 : public Derived
36
     {
37
     public:
38
          D2(int z): Derived(z)
39
40
              std::cout << "D2()\n";
41
          }
42
          ~D2()
43
          {
44
              std::cout << "~D2()\n";
45
          }
46
47
              // note: no print() function here
48
     };
49
50
     int main()
51
52
          D2 d(5);
53
          d.print();
54
     }
```

#### **Hide Solution**

Base()
Derived()
D2()
Derived: 5
~D2()
~Derived()
~Base()

2a) Write an Apple class and a Banana class that are derived from a common Fruit class. Fruit should have two members: a name, and a color.

The following program should run:

```
int main()
{
    Apple a("red");
    Banana b;

std::cout << "My " << a.getName() << " is " << a.getColor() << ".\n";</pre>
```

```
7    std::cout << "My " << b.getName() << " is " << b.getColor() << ".\n";
8    return 0;
10 }</pre>
```

And produce the result:

```
My apple is red.
My banana is yellow.
```

#### **Hide Solution**

```
#include <iostream>
2
     #include <string>
3
4
     class Fruit
5
     {
6
     private:
7
         std::string m_name;
8
         std::string m_color;
9
10
     public:
11
          Fruit(std::string name, std::string color)
12
              : m_name(name), m_color(color)
13
         {
14
15
     }
16
17
         std::string getName() { return m_name; }
18
          std::string getColor() { return m_color; }
19
     };
20
21
     class Apple: public Fruit
22
     {
23
     public:
24
         Apple(std::string color="red")
25
            : Fruit("apple", color)
26
27
         }
28
     };
29
30
     class Banana : public Fruit
31
     {
32
     public:
33
         Banana()
34
             : Fruit("banana", "yellow")
35
37
         }
38
     };
39
40
     int main()
41
42
         Apple a("red");
43
         Banana b;
44
         std::cout << "My " << a.getName() << " is " << a.getColor() << ".\n";
45
         std::cout << "My " << b.getName() << " is " << b.getColor() << ".\n";
46
47
48
         return 0;
49
     }
```

2b) Add a new class to the previous program called GrannySmith that inherits from Apple.

The following program should run:

```
int main()
 2
 3
          Apple a("red");
 4
          Banana b;
 5
          GrannySmith c;
 6
 7
          std::cout << "My " << a.getName() << " is " << a.getColor() << ".\n";
 8
          std::cout << "My " << b.getName() << " is " << b.getColor() << ".\n";
 9
          std::cout << "My " << c.getName() << " is " << c.getColor() << ".\n";
10
11
          return 0;
12
And produce the result:
My apple is red.
```

#### **Hide Solution**

My banana is yellow.

My granny smith apple is green.

```
#include <iostream>
2
     #include <string>
3
4
     class Fruit
5
     {
6
     private:
7
         std::string m_name;
8
         std::string m_color;
9
     public:
10
         Fruit(std::string name, std::string color)
11
12
              : m_name(name), m_color(color)
13
14
15
         }
16
17
         std::string getName() { return m_name; }
18
         std::string getColor() { return m_color; }
19
     };
20
21
     class Apple: public Fruit
22
23
     // The previous constructor we used for Apple had a fixed name ("apple").
24
     // We need a new constructor for GrannySmith to use to set the name of the fruit
25
     protected: // protected so only derived classes can access
26
         Apple(std::string name, std::string color)
27
             : Fruit(name, color)
28
29
         }
30
31
     public:
32
         Apple(std::string color="red")
33
            : Fruit("apple", color)
34
         {
35
         }
36
     };
37
38
     class Banana : public Fruit
39
     {
40
     public:
41
42
              : Fruit("banana", "yellow")
43
         {
44
45
         }
```

```
46
     };
47
48
     class GrannySmith : public Apple
49
50
     public:
         GrannySmith()
51
52
              : Apple("granny smith apple", "green")
53
         {
54
55
         }
56
     };
57
58
     int main()
59
60
         Apple a("red");
61
         Banana b;
         GrannySmith c;
63
64
         std::cout << "My " << a.getName() << " is " << a.getColor() << ".\n";
65
         std::cout << "My " << b.getName() << " is " << b.getColor() << ".\n";
         std::cout << "My " << c.getName() << " is " << c.getColor() << ".\n";
67
68
         return 0;
69
```

3) Challenge time! The following quiz question is more difficult and lengthy. We're going to write a simple game where you fight monsters. The goal of the game is to collect as much gold as you can before you die or get to level 20.

Our program is going to consist of 3 classes: A Creature class, a Player class, and a Monster class. Player and Monster both inherit from Creature.

3a) First create the Creature class. Creatures have 5 attributes: A name (std::string), a symbol (a char), an amount of health (int), the amount of damage they do per attack (int), and the amount of gold they are carrying (int). Implement these as class members. Write a full set of getters (a get function for each member). Add three other functions: void reduceHealth(int) reduces the Creature's health by an integer amount. bool isDead() returns true when the Creature's health is 0 or less. void addGold(int) adds gold to the Creature.

The following program should run:

```
1
     #include <iostream>
2
     #include <string>
3
4
     int main()
5
     {
         Creature o("orc", 'o', 4, 2, 10);
6
7
         o.addGold(5);
8
         o.reduceHealth(1);
         std::cout << "The " << o.getName() << " has " << o.getHealth() << " health and is carrying " << o.g</pre>
9
10
     etGold() << " gold.\n.";
11
12
         return 0;
     }
```

And produce the result:

The orc has 3 health and is carrying 15 gold.

```
#include <iostream>
#include <string>

class Creature

{
protected:
    std::string m_name;
    char m_symbol;
```

```
9
         int m_health;
10
         int m_damage;
11
         int m_gold;
12
13
     public:
14
         Creature(std::string name, char symbol, int health, int damage, int gold) :
15
              m_name(name), m_symbol(symbol), m_health(health), m_damage(damage), m_gold(gold)
16
          {
         }
17
18
19
         const std::string& getName() { return m_name; }
20
          char getSymbol() { return m_symbol; }
21
         int getHealth() { return m_health; }
         int getDamage() { return m_damage; }
23
         int getGold() { return m_gold; }
24
25
         void reduceHealth(int health) { m_health -= health; }
26
         bool isDead() { return m_health <= 0; }</pre>
27
         void addGold(int gold) { m_gold += gold; }
28
     };
29
30
     int main()
31
32
         Creature o("orc", 'o', 4, 2, 10);
33
         o.addGold(5);
         o.reduceHealth(1);
34
          std::cout << "The " << o.getName() << " has " << o.getHealth() << " health and is carrying " << o.g
35
     etGold() << " gold.\n.";
37
38
         return 0;
```

3b) Now we're going to create the Player class. The Player class inherits from Creature. Player has one additional member, the player's level, which starts at 1. The player has a custom name (entered by the user), uses symbol '@', has 10 health, does 1 damage to start, and has no gold. Write a function called levelUp() that increases the player's level and damage by 1. Also write a getter for the level member. Finally, write a function called hasWon() that returns true if the player has reached level 20.

Write a new main() function that asks the user for their name and produces the output as follows:

```
Enter your name: Alex Welcome, Alex.
You have 10 health and are carrying 0 gold.
```

```
#include <iostream>
1
2
     #include <string>
3
4
     class Creature
5
6
     protected:
7
         std::string m_name;
8
         char m_symbol;
9
         int m_health;
10
         int m_damage;
11
         int m_gold;
12
13
     public:
14
         Creature(std::string name, char symbol, int health, int damage, int gold) :
15
             m_name(name), m_symbol(symbol), m_health(health), m_damage(damage), m_gold(gold)
16
         {
         }
17
18
19
         const std::string& getName() { return m_name; }
20
         char getSymbol() { return m_symbol; }
```

```
21
         int getHealth() { return m_health; }
22
         int getDamage() { return m_damage; }
23
         int getGold() { return m_gold; }
24
25
         void reduceHealth(int health) { m_health -= health; }
26
         bool isDead() { return m_health <= 0; }</pre>
27
         void addGold(int gold) { m_gold += gold; }
28
     };
29
     class Player : public Creature
30
31
32
         int m_level = 1;
33
34
     public:
35
         Player(std::string name)
36
            : Creature(name, '@', 10, 1, 0)
37
38
         }
39
40
         void levelUp()
41
42
              ++m_level;
43
              ++m_damage;
44
45
46
         int getLevel() { return m_level; }
47
          bool hasWon() { return m_level >= 20; }
48
     };
49
50
     int main()
51
52
         std::cout << "Enter your name: ";</pre>
53
         std::string playerName;
54
         std::cin >> playerName;
55
56
         Player p(playerName);
57
         std::cout << "Welcome, " << p.getName() << ".\n";</pre>
58
59
         std::cout << "You have " << p.getHealth() << " health and are carrying " << p.getGold() << " gol</pre>
60
     d.\n.";
61
62
         return 0;
```

3c) Next up is the Monster class. Monster also inherits from Creature. Monsters have no non-inherited member variables.

First, write an empty Monster class inheriting from Creature, and then add an enum inside the Monster class named Type that contains enumerators for the 3 monsters that we'll have in this game: DRAGON, ORC, and SLIME (you'll also want a MAX\_TYPES enumerator, as that will come in handy in a bit).

# **Hide Solution**

```
1
     class Monster : public Creature
2
     {
3
     public:
4
          enum Type
5
6
              DRAGON,
7
              ORC,
8
              SLIME,
9
              MAX_TYPES
10
          };
    };
```

3d) Each Monster type will have a different name, symbol, starting health, gold, and damage. Here is a table of stats for each monster Type:

Туре	Name	Symbol	Health	Damage	Gold
DRAGON	dragon	D	20	4	100
ORC	orc	0	4	2	25
SLIME	slime	S	1	1	10

Next step is to write a Monster constructor, so we can create monsters. The Monster constructor should take a Type enum as a parameter, and then create a Monster with the appropriate stats for that kind of monster.

There are a number of different ways to implement this (some better, some worse). However in this case, because all of our monster attributes are predefined (not random), we'll use a lookup table. A lookup table is an array that holds all of the predefined attributes. We can use the lookup table to look up the attributes for a given monster as needed.

So how do we implement this lookup table? It's not hard. We just need two things. First, we need an array that contains an element for each monster Type. Each array element will contain a struct that contains all of the predefined attribute values for that Type of Monster.

Step 1: Create a struct type inside Monster named MonsterData. This struct should have a member for each attribute (name, symbol, health, damage, and gold).

Step 2: Declare an array of that struct as a static member of the class named monsterData (declare this like a normal array member, and then add the word static before it).

Step 3: Add the following code outside of the class. This is the definition for our lookup table:

Now we can index this array to lookup any values we need! For example, to get a Dragon's gold, we can access monsterData[DRAGON].gold.

Use this lookup table to implement your constructor:

```
1 Monster(Type type): Creature(monsterData[type].name, ...)
```

The following program should compile:

```
#include <iostream>
#include <string>

int main()

{
    Monster m(Monster::ORC);
    std::cout << "A " << m.getName() << " (" << m.getSymbol() << ") was created.\n";
}</pre>
```

and print:

A orc (o) was created.

```
1
     #include <iostream>
2
     #include <string>
3
4
     class Creature
5
     {
6
     protected:
7
         std::string m_name;
8
         char m_symbol;
9
         int m_health;
         int m_damage;
```

```
11
         int m_gold;
12
13
     public:
14
         Creature(std::string name, char symbol, int health, int damage, int gold) :
15
             m_name(name), m_symbol(symbol), m_health(health), m_damage(damage), m_gold(gold)
16
          {
17
         }
18
19
         const std::string& getName() { return m_name; }
20
          char getSymbol() { return m_symbol; }
21
         int getHealth() { return m_health; }
22
         int getDamage() { return m_damage; }
23
         int getGold() { return m_gold; }
24
25
         void reduceHealth(int health) { m_health -= health; }
26
         bool isDead() { return m_health <= 0; }</pre>
27
         void addGold(int gold) { m_gold += gold; }
28
     };
29
30
     class Player : public Creature
31
32
         int m_level = 1;
33
34
     public:
35
         Player(std::string name)
36
              : Creature(name, '@', 10, 1, 0)
37
38
         }
39
40
         void levelUp()
41
42
             ++m_level;
43
             ++m_damage;
44
         }
45
         int getLevel() { return m_level; }
46
47
     };
48
49
     class Monster : public Creature
50
     {
51
     public:
52
         enum Type
53
54
             DRAGON,
55
             ORC,
56
              SLIME,
57
             MAX_TYPES
58
         };
59
60
         struct MonsterData
61
62
              const char* name;
63
             char symbol;
64
              int health;
65
             int damage;
66
              int gold;
67
         };
68
69
         static MonsterData monsterData[MAX_TYPES];
70
71
         Monster(Type type)
72
              : Creature(monsterData[type].name, monsterData[type].symbol, monsterData[type].health, monsterD
73
     ata[type].damage, monsterData[type].gold)
74
         {
75
76
     };
77
```

```
78
     Monster::MonsterData Monster::monsterData[Monster::MAX_TYPES]
79
     {
80
         { "dragon", 'D', 20, 4, 100 },
81
         { "orc", 'o', 4, 2, 25 },
         { "slime", 's', 1, 1, 10 }
82
83
     };
84
85
     int main()
86
87
         Monster m(Monster::ORC);
88
         std::cout << "A " << m.getName() << " (" << m.getSymbol() << ") was created.\n";
89
90
         return 0;
```

3e) Finally, add a static function to Monster named getRandomMonster(). This function should pick a random number between 0 and MAX\_TYPES-1 and return a monster (by value) with that Type (you'll need to static\_cast the int to a Type to pass it to the Monster constructor).

You can use the following code to pick a random number:

```
1
     #include <cstdlib> // for rand() and srand()
2
     #include <ctime> // for time()
3
     // Generate a random number between min and max (inclusive)
4
     // Assumes srand() has already been called
5
     int getRandomNumber(int min, int max)
6
7
     {
8
         static const double fraction = 1.0 / (static_cast<double>(RAND_MAX) + 1.0); // static used for eff
9
     iciency, so we only calculate this value once
         // evenly distribute the random number across our range
10
         return static_cast<int>(rand() * fraction * (max - min + 1) + min);
11
     }
```

The following main function should run:

```
1
     #include <iostream>
2
     #include <string>
 3
     #include <cstdlib> // for rand() and srand()
4
     #include <ctime> // for time()
5
6
     int main()
 7
          srand(static_cast<unsigned int>(time(0))); // set initial seed value to system clock
8
9
          rand(); // get rid of first result
10
11
          for (int i = 0; i < 10; ++i)
12
13
             Monster m = Monster::getRandomMonster();
              std::cout << "A " << m.getName() << " (" << m.getSymbol() << ") was created.\n";
14
15
16
17
         return 0;
18
```

The results of this program should be randomized.

```
#include <iostream>
#include <string>
#include <cstdlib> // for rand() and srand()
#include <ctime> // for time()

// Generate a random number between min and max (inclusive)
// Assumes srand() has already been called
int getRandomNumber(int min, int max)
```

```
9
10
          static const double fraction = 1.0 / (static_cast<double>(RAND_MAX) + 1.0); // static used for ef
11
      ficiency, so we only calculate this value once
12
                                                                                          // evenly distribute
       the random number across our range
13
14
          return static_cast<int>(rand() * fraction * (max - min + 1) + min);
15
      }
16
17
      class Creature
18
      {
19
      protected:
20
          std::string m_name;
21
          char m_symbol;
22
          int m_health;
23
          int m_damage;
24
          int m_gold;
25
26
      public:
27
          Creature(std::string name, char symbol, int health, int damage, int gold) :
28
              m_name(name), m_symbol(symbol), m_health(health), m_damage(damage), m_gold(gold)
29
          {
30
          }
31
32
          char getSymbol() { return m_symbol; }
33
          const std::string& getName() { return m_name; }
34
          bool isDead() { return m_health <= 0; }</pre>
35
          int getGold() { return m_gold; }
          void addGold(int gold) { m_gold += gold; }
36
37
          void reduceHealth(int health) { m_health -= health; }
38
          int getHealth() { return m_health; }
39
          int getDamage() { return m_damage; }
40
     };
41
42
      class Player : public Creature
43
44
      int m_level = 1;
45
46
      public:
47
          Player(std::string name)
              : Creature(name, '@', 10, 1, 0)
48
49
50
          }
51
52
          void levelUp()
53
          {
54
              ++m_level;
55
              ++m_damage;
56
57
58
          int getLevel() { return m_level; }
59
          bool hasWon() { return m_level >= 20; }
60
     };
61
62
      class Monster : public Creature
63
      {
64
      public:
65
          enum Type
66
67
              DRAGON,
68
              ORC,
              SLIME,
69
70
              MAX_TYPES
71
          };
72
73
          struct MonsterData
74
75
              const char* name;
```

```
char symbol;
76
77
              int health;
78
              int damage;
79
              int gold;
80
          };
81
82
          static MonsterData monsterData[MAX_TYPES];
83
84
          Monster(Type type)
85
               : Creature(monsterData[type].name, monsterData[type].symbol, monsterData[type].health, monster
86
      Data[type].damage, monsterData[type].gold)
87
88
89
90
          static Monster getRandomMonster()
91
92
              int num = getRandomNumber(0, MAX_TYPES - 1);
93
              return Monster(static_cast<Type>(num));
94
          }
95
      };
96
97
      Monster::MonsterData Monster::monsterData[Monster::MAX_TYPES]
98
          { "dragon", 'D', 20, 4, 100 },
99
          { "orc", 'o', 4, 2, 25 },
100
          { "slime", 's', 1, 1, 10 }
101
102
      };
103
104
      int main()
105
106
          srand(static_cast<unsigned int>(time(0))); // set initial seed value to system clock
107
          rand(); // get rid of first result
108
109
          for (int i = 0; i < 10; ++i)
110
111
              Monster m = Monster::getRandomMonster();
112
              std::cout << "A " << m.getName() << " (" << m.getSymbol() << ") was created.\n";
113
          return 0;
      }
```

3f) We're finally set to write our game logic!

Here are the rules for the game:

- The player encounters one randomly generated monster at a time.
- For each monster, the player has two choices: (R)un or (F)ight.
- If the player decides to Run, they have a 50% chance of escaping.
- If the player escapes, they move to the next encounter will no ill effects.
- If the player does not escape, the monster gets a free attack, and the player chooses their next action.
- If the player chooses to fight, the player attacks first. The monster's health is reduced by the player's damage.
- If the monster dies, the player takes any gold the monster is carrying. The player also levels up, increasing their level and damage by 1.
- If the monster does not die, the monster attacks the player back. The player's health is reduced by the monster's damage.
- The game ends when the player has died (loss) or reached level 20 (win)
- If the player dies, the game should tell the player what level they were and how much gold they had.
- · If the player wins, the game should tell the player they won, and how much gold they had

Here's a sample game session:

```
Enter your name: Alex
Welcome, Alex
You have encountered a slime (s).
(R)un or (F)ight: f
```

```
You hit the slime for 1 damage.
You killed the slime.
You are now level 2.
You found 10 gold.
You have encountered a dragon (D).
(R)un or (F)ight: r
You failed to flee.
The dragon hit you for 4 damage.
(R)un or (F)ight: r
You successfully fled.
You have encountered a orc (o).
(R)un or (F)ight: f
You hit the orc for 2 damage.
The orc hit you for 2 damage.
(R)un or (F)ight: f
You hit the orc for 2 damage.
The orc hit you for 2 damage.
(R)un or (F)ight: f
You hit the orc for 2 damage.
You are now level 3.
You found 25 gold.
You have encountered a dragon (D).
(R)un or (F)ight: r
You failed to flee.
The dragon hit you for 4 damage.
You died at level 3 and with 35 gold.
Too bad you can't take it with you!
```

#### Hint: Create 4 functions:

- 1. The main() function should handle game setup (creating the Player) and the main game loop.
- 2. fightMonster() handles the fight between the Player and a single Monster, including asking the player what they want to do, handling the run or fight cases.
- 3. attackMonster() handles the player attacking the monster, including leveling up.
- 4. attackPlayer() handles the monster attacking the player.

```
#include <iostream>
1
2
     #include <string>
3
     #include <cstdlib> // for rand() and srand()
4
     #include <ctime> // for time()
5
     // Generate a random number between min and max (inclusive)
6
     // Assumes srand() has already been called
7
8
     int getRandomNumber(int min, int max)
9
          static const double fraction = 1.0 / (static_cast<double>(RAND_MAX) + 1.0); // static used for ef
10
11
      ficiency, so we only calculate this value once
12
                                                                                        // evenly distribute
13
      the random number across our range
14
          return static_cast<int>(rand() * fraction * (max - min + 1) + min);
15
     }
16
17
     class Creature
18
19
     protected:
20
          std::string m_name;
21
          char m_symbol;
22
          int m_health;
23
         int m_damage;
24
          int m_gold;
```

```
26
      public:
27
          Creature(std::string name, char symbol, int health, int damage, int gold) :
28
              m_name(name), m_symbol(symbol), m_health(health), m_damage(damage), m_gold(gold)
29
          }
30
31
32
          char getSymbol() { return m_symbol; }
33
          const std::string& getName() { return m_name; }
34
          bool isDead() { return m_health <= 0; }</pre>
35
          int getGold() { return m_gold; }
36
          void addGold(int gold) { m_gold += gold; }
37
          void reduceHealth(int health) { m_health -= health; }
38
          int getHealth() { return m_health; }
39
          int getDamage() { return m_damage; }
40
      };
41
42
      class Player : public Creature
43
      {
44
          int m_level = 1;
45
46
      public:
47
          Player(std::string name)
48
              : Creature(name, '@', 10, 1, 0)
49
          {
          }
50
51
52
          void levelUp()
53
54
              ++m_level;
55
              ++m_damage;
56
          }
57
58
          int getLevel() { return m_level; }
59
          bool hasWon() { return m_level >= 20; }
60
      };
61
62
63
64
      class Monster : public Creature
65
      {
66
      public:
67
          enum Type
68
69
              DRAGON,
              ORC,
70
71
              SLIME,
72
              MAX_TYPES
73
          };
74
75
          struct MonsterData
76
          {
77
              const char* name;
78
              char symbol;
79
              int health;
80
              int damage;
81
              int gold;
82
          };
83
84
          static MonsterData monsterData[MAX_TYPES];
85
          Type m_type;
86
87
          Monster(Type type)
              : Creature(monsterData[type].name, monsterData[type].symbol, monsterData[type].health, monster
88
89
      Data[type].damage, monsterData[type].gold), m_type(type)
90
          {
91
          }
```

```
93
          static Monster getRandomMonster()
94
95
               int num = getRandomNumber(0, MAX_TYPES - 1);
96
               return Monster(static_cast<Type>(num));
97
          }
98
      };
99
100
      Monster::MonsterData Monster::monsterData[Monster::MAX_TYPES]
101
102
          { "dragon", 'D', 20, 4, 100 },
          { "orc", 'o', 6, 2, 25 },
104
          { "slime", 's', 1, 1, 5 }
105
      };
106
107
      // This function handles the player attacking the monster
108
      void attackMonster(Player &p, Monster &m)
109
110
          // If the player is dead, we can't attack the monster
111
          if (p.isDead())
112
              return;
113
          std::cout << "You hit the " << m.getName() << " for " << p.getDamage() << " damage.\n";</pre>
114
115
116
          // Reduce the monster's health by the player's damage
117
          m.reduceHealth(p.getDamage());
118
119
          // If the monster is now dead, level the player up
120
          if (m.isDead())
121
          {
               std::cout << "You killed the " << m.getName() << ".\n";</pre>
122
123
               p.levelUp();
124
              std::cout << "You are now level " << p.getLevel() << ".\n";</pre>
125
               std::cout << "You found " << m.getGold() << " gold.\n";</pre>
126
              p.addGold(m.getGold());
127
128
      }
129
130
      // This function handles the monster attacking the player
131
      void attackPlayer(Monster &m, Player &p)
132
133
          // If the monster is dead, it can't attack the player
134
          if (m.isDead())
135
              return;
136
          // Reduce the player's health by the monster's damage
137
138
          p.reduceHealth(m.getDamage());
139
          std::cout << "The " << m.getName() << " hit you for " << m.getDamage() << " damage.\n";</pre>
140
      }
141
142
      // This function handles the entire fight between a player and a randomly generated monster
143
      void fightMonster(Player &p)
144
      {
145
          // First randomly generate a monster
146
          Monster m = Monster::getRandomMonster();
          std::cout << "You have encountered a " << m.getName() << " (" << m.getSymbol() << ").\n";</pre>
147
148
149
          // While the monster isn't dead and the player isn't dead, the fight continues
150
          while (!m.isDead() && !p.isDead())
151
          {
152
              std::cout << "(R)un or (F)ight: ";</pre>
153
              char input;
              std::cin >> input;
154
155
              if (input == 'R' || input == 'r')
156
               {
157
                   // 50% chance of fleeing successfully
158
                   if (getRandomNumber(1, 2) == 1)
```

```
159
                   {
160
                       std::cout << "You successfully fled.\n";</pre>
161
                       return; // success ends the encounter
162
                   }
163
                   else
164
                   {
165
                       // Failure to flee gives the monster a free attack on the player
                       std::cout << "You failed to flee.\n";</pre>
166
                       attackPlayer(m, p);
167
168
                       continue;
169
                   }
170
171
172
               if (input == 'F' || input == 'f')
173
174
                   // Player attacks first, monster attacks second
175
                   attackMonster(p, m);
176
                   attackPlayer(m, p);
177
178
          }
179
      }
180
181
      int main()
182
      {
           srand(static_cast<unsigned int>(time(0))); // set initial seed value to system clock
183
184
           rand(); // get rid of first result
185
186
           std::cout << "Enter your name: ";</pre>
187
           std::string playerName;
188
           std::cin >> playerName;
189
190
           Player p(playerName);
191
           std::cout << "Welcome, " << p.getName() << '\n';</pre>
192
193
           // If the player isn't dead and hasn't won yet, the game continues
194
          while (!p.isDead() && !p.hasWon())
195
               fightMonster(p);
196
197
           // At this point, the player is either dead or has won
198
          if (p.isDead())
199
200
               std::cout << "You died at level " << p.getLevel() << " and with " << p.getGold() << " gold.\n"
201
202
               std::cout << "Too bad you can't take it with you!\n";</pre>
203
          }
204
           else
206
               std::cout << "You won the game with " << p.getGold() << " gold!\n";</pre>
           return 0;
```



### 12.1 -- Pointers and references to the base class of derived objects



<u>Index</u>



11.7 -- Multiple inheritance