

Learning Objectives: Mutability

- **Define the term mutable**
- **Construct an external function to modify class variables (attributes)**

Mutability and External Functions

Mutability

Objects are mutable, which means that objects (specifically their attributes) can change value. Think of a video game; the main character in the game is constantly changing. It could be their position on the screen, the score, their health, the items in their inventory, etc. Imagine a simple class called `Player`. A newly instantiated `Player` object has a health of 100, a score of 0, and starts on level 1. This object can lose health, increase their score, and advance levels.

```
//add class definitions below this line
```

```
class Player {  
    public:  
        int health;  
        int score;  
        int level;  
  
    Player() {  
        health = 100;  
        score = 0;  
        level = 1;  
    }  
};
```

```
//add class definitions above this line
```

Print out the attributes of `player1`. Then change each attribute and print out the attributes again.

//add code below this line

```
Player player1;
cout << "This player has " << player1.health << " health, a
      score of " << player1.score;
cout << ", and is on level " << player1.level << "." << endl;
player1.health -= 10;
player1.score += 25;
player1.level += 1;
cout << "This player has " << player1.health << " health, a
      score of " << player1.score;
cout << ", and is on level " << player1.level << "." << endl;
```

//add code above this line

challenge

Try these variations:

- Change the health of player1 to 0.
- Print the status of player1 specifying that their health is 0 and the message Game over..

▼ One Possible Solution

```
player1.health = 0;
cout << "This player has " << player1.health << " health.
      Game over." << endl;
```

External Functions and Objects

One of the benefits of functions is code reusability. The example above has a repetition of the cout statement. This is a good opportunity to use a function.

//add function definitions below this line

```
void PrintPlayer(Player p) {
    cout << "This player has " << p.health << " health, a score of
          " << p.score;
    cout << ", and is on level " << p.level << "." << endl;
}
```

//add function definitions above this line

In the main function, replace the strings inside the print statements with a call to the PrintPlayer function. Don't forget to pass the player1 object to the PrintPlayer function.

```
//add code below this line
```

```
Player player1;  
PrintPlayer(player1);  
player1.health -= 10;  
player1.score += 25;  
player1.level += 1;  
PrintPlayer(player1);
```

```
//add code above this line
```

Using an external function to print the status of player1 may not seem like it was worth the effort to change the code. But when these functions become more complex, the efficiency becomes clear.

```
//add function definitions below this line
```

```
void PrintPlayer(Player p) {  
    if (p.health <= 0) {  
        cout << "This player is dead. They died on level " <<  
            p.level;  
        cout << " with a score of " << p.score << "." << endl;  
    }  
    else {  
        cout << "This player has " << p.health << " health, a score  
            of " << p.score;  
        cout << ", and is on level " << p.level << "." << endl;  
    }  
}
```

```
//add function definitions above this line
```

Now that the PrintPlayer function can return two different strings, call the function when the player1 object has full health, and call it again when the object has no health.

```
//add code below this line
```

```
Player player1;  
PrintPlayer(player1);  
player1.health = 0;  
player1.score += 25;  
player1.level += 1;  
PrintPlayer(player1);
```

```
//add code above this line
```

The problem with using external functions, however, is that the changes made to objects in one function do not translate or carry over to the next function.

```
#include <iostream>  
using namespace std;  
  
//add class definitions below this line  
  
class Player {  
public:  
    int health;  
    int score;  
    int level;  
  
    Player() {  
        health = 100;  
        score = 0;  
        level = 1;  
    }  
};  
  
//add class definitions above this line  
  
//add function definitions below this line  
  
void PrintPlayer(Player p) {  
    if (p.health <= 0) {  
        cout << "This player is dead. They died on level " <<  
p.level;  
        cout << " with a score of " << p.score << "." << endl;  
    }  
    else {  
        cout << "This player has " << p.health << " health, a score  
of " << p.score;
```

```

        cout << ", and is on level " << p.level << "." << endl;
    }
}

void ChangeHealth(Player p, int amount) {
    p.health += amount;
    cout << "New health = " << p.health << endl;
}

//add function definitions above this line

int main() {

    //add code below this line

    Player player1;
    PrintPlayer(player1);
    player1.health = 0;
    player1.score += 25;
    player1.level += 1;
    PrintPlayer(player1);

    ChangeHealth(player1, 20); //changes health of player1
    PrintPlayer(player1); //does not register changes from
    ChangeHealth function

    //add code above this line

    return 0;

}

```

You'll notice above that `ChangeHealth(player1, 20)` had no effect on `PrintPlayer(player1)`. `player1`'s health changed to 20 but after it leaves the function, it returned to 0. Changes that occur within external functions are not permanent. Next, you will be introduced to **class functions** which will enable you to make changes to objects that are more permanent.