Learning Objectives: Class Functions

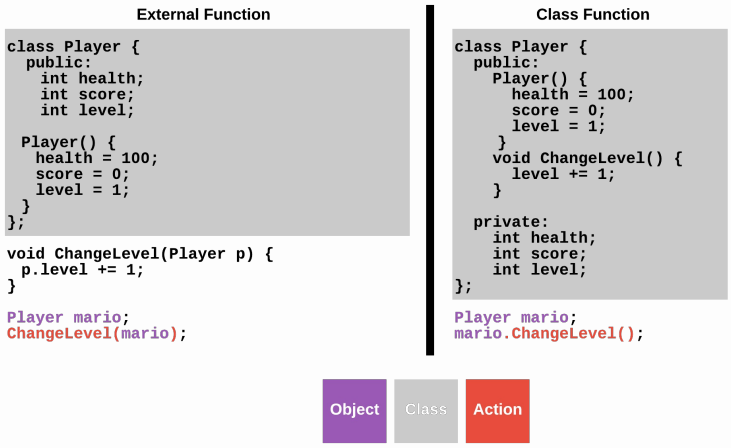
- **Define the term class function**
- **Convert an external function that modifies an object into a class function**
- **Demonstrate the syntax for defining and calling a class function**

External Functions vs. Class Functions

Class Functions

Back in the *Introduction to Objects* module, a class was defined as "a collection of data and the actions that can modify the data." The constructor can build the "collection of data", but nothing in the class can modify the data. Instead, external functions were used to modify the object. However, using external functions to modify objects is not a good practice. Here, you will be introduced to **class functions**, also known as *class or instance methods*, that serve to modify the data within objects.

Think of a class function as a function that is attached to an object. The class function is the most common type of function when creating classes. Notice how class functions are declared *inside* of the class. These functions are called class functions because they have access to the class variables (the attributes declared in the constructor). Class functions are invoked using dot-notation.



.guides/img/mutability/ExternalVsClassFunctions

There are a few notable differences between **external functions** and **class functions**.
1. In order for external functions to work, class variables within a class must be `public`. Otherwise, the function will not be able to act on the object. This, however, is not a good practice.
1. In C++, it is a best practice to make class variables `private`. Doing so prevents external functions from accidentally making undesirable changes to the class attributes. This is why class functions are preferred when

modifying objects.
1. Everything within the class is `public` in the external function example. In contrast, class attributes are `private` while constructors and functions are `public` in the class function example.
1. To modify an object using an external function, the syntax is `Function(Object)` (i.e. `ChangeLevel(mario)`). On the other hand, the syntax for using a class function is `Object.Function()` (i.e. `mario.ChangeLevel()`).

Converting to Class Functions

When mutability was first introduced, you made a `Player` class with a few external functions. You are now going to transform these external functions into class functions moving forward. The `Player` class will be defined just as before. This time, however, `PrintPlayer` will be a part of the class.

```
//add class definitions below this line

class Player {
  public: //public access modifer
    Player() { //constructor
      health = 100;
      score = 0;
      level = 1;
    }
    void PrintPlayer() { //class function
      if (health <= 0) {
        cout << "This player is dead. They died on level " <<
        level;
        cout << " with a score of " << score << "." << endl;
      }
      else {
        cout << "This player has " << health << " health, a
        score of " << score;
        cout << ", and is on level " << level << "." << endl;
      }
    }

  private: //private access modifiers
    int health;
    int score;
    int level;
};

//add class definitions above this line
```

In `main`, instantiate a `Player` object. Then call the class function `PrintPlayer` using dot-notation. Be sure to label the appropriate `public` and `private` access modifiers!

```
  //add code below this line

  Player mario;
  mario.PrintPlayer();

  //add code above this line
```

challenge

Try this variation:

Call `PrintPlayer` like this:

```
Player mario;
PrintPlayer(mario);
```

▼ **Why did this generate an error?**
C++ says that `'PrintPlayer' was not declared in this scope` even though the definition is within the class `Player`. This happens because the code is using an external function call. Currently, `PrintPlayer` is a class function within the `Player` class. In order to access a class function, dot notation must be used.

More Player Methods

The next class functions to add to the `Player` class are those to print the `health` and `level` attributes of the `Player` object. Start with the class function `ChangeHealth`. This function takes `amount` as an int parameter. `ChangeHealth` will add `amount` to the `health` attribute. If a player's health increases, `amount` is positive. If their health decreases, `amount` is negative. Add `ChangeHealth` directly below `PrintPlayer()`.

```
void PrintPlayer() { //class function
  if (health <= 0) {
    cout << "This player is dead. They died on level " << level;
    cout << " with a score of " << score << "." << endl;
  }
  else {
    cout << "This player has " << health << " health, a score of
        " << score;
    cout << ", and is on level " << level << "." << endl;
  }
}
void ChangeHealth(int amount) {
  health += amount;
}
```

The class function `NextLevel` is going to be similar to `ChangeHealth` except for one difference. `NextLevel` has no parameters. In video games, players go up in levels; rarely do they decrease. So the `level` attribute will increase by one when the class function `NextLevel` is called.

```
void ChangeHealth(int amount) {
  health += amount;
}
void NextLevel() {
  level++;
}
```

Change `main` to:

```
  //add code below this line

  Player mario;
  mario.PrintPlayer();
  mario.ChangeHealth(25);
  mario.NextLevel();
  mario.PrintPlayer();

  //add code above this line
```

Then `TRY IT`.

challenge

Try these variations:

- Change `mario.ChangeHealth(25);` to `mario.ChangeHealth(-25);`.
- Add another `mario.NextLevel();` below the first `mario.NextLevel();`.
- Create a class function to change a player's `score` by a set integer `amount`. Then try to call it.

  ▼ **One possible solution**

```
//class function
void ChangeScore(int amount) {
  score += amount;
}
```

```
//main
mario.ChangeScore(123);
```

▼ **Why learn about external functions that modify objects when C++ has class functions?**

It might seem like a waste of time to learn how to write external functions that modify objects, but this approach builds upon concepts you have already seen — external functions and objects. This allows you to understand mutability without having to worry about class functions. Once you understand how these ideas work, transforming an external function into an class function is much simpler. External functions that modify objects serve as an intermediary step on the way to learning about class functions.

More Class Functions

More on Class Methods and Objects

Changes to objects should happen exclusively through class functions. This makes your code easier to organize and easier for others to understand. Imagine you are going to create a class that keeps track of a meal. In this case, a meal can be thought of as all of the drinks, appetizers, courses, and desserts served. Each one of these categories will become a class variable (attribute). Assign each attribute a vector of strings. Remember, class variables/attribute are `private`.

```
//add class definitions below this line

class Meal {
  private:
    vector<string> drinks;
    vector<string> appetizers;
    vector<string> main_courses;
    vector<string> desserts;
};

//add class definitions above this line
```

Next, add a class function to add a drink to the `Meal` object. Use the `push_back` function to add an element to the vector. So `drinks.push_back(drink)` adds the drink `drink` to the vector `drinks`. Then add a class function `PrintDrinks` to print out all of the elements inside the `drinks` vector. Class functions are `public`.

```
//add class definitions below this line

class Meal {
  public:
    void AddDrink(string drink) {
      drinks.push_back(drink);
    }
    void PrintDrinks() {
      for (auto a: drinks) {
        cout << a << endl;
      }
    }

  private:
    vector<string> drinks;
    vector<string> appetizers;
    vector<string> main_courses;
    vector<string> desserts;
};

//add class definitions above this line
```

Create a `Meal` object in `main` and then test your code with the following added commands.

```
  //add code below this line

  Meal dinner;
  dinner.AddDrink("water");
  dinner.PrintDrinks();

  //add code above this line
```

Now create the `AddAppetizer` class function for the class. Like the `AddDrink` function above, `AddAppetizer` accepts a string as a parameter and adds it as an element to the `appetizers` attribute (which is a vector). Then create a `PrintAppetizers` function to print what's inside the `appetizers` vector.

```
void AddDrink(string drink) {
  drinks.push_back(drink);
}
void PrintDrinks() {
  for (auto a: drinks) {
    cout << a << endl;
  }
}
void AddAppetizer(string app) {
  appetizers.push_back(app);
}
void PrintAppetizers() {
  for (auto a: appetizers) {
    cout << a << endl;
  }
}
```

Add `"bruschetta"` as an appetizer to the `dinner` object, then call the class function `PrintAppetizers` like below.

```
  //add code below this line

  Meal dinner;
  dinner.AddDrink("water");
  dinner.PrintDrinks();
  dinner.AddAppetizer("bruschetta");
  dinner.PrintAppetizers();

  //add code above this line
```

challenge

Create the following class functions:

- `AddMainCourse` - accepts a string which represents a main course and adds it to the meal.
- `PrintMainCourses` - prints all of the main courses in the meal.
- `AddDessert` - accepts a string which represents a dessert and adds it to the meal.
- `PrintDesserts` - prints all of the desserts in the meal.

- Test your code using `"roast chicken"` as a main course and `"chocolate cake"` as a dessert. Then use the `Print` class functions you created to print out all of the items of the meal.

  ▼

  Meal code

```
#include <iostream>
#include <vector>
using namespace std;

//add class definitions below this line

class Meal {
  public:
    void AddDrink(string drink) {
      drinks.push_back(drink);
    }
    void PrintDrinks() {
      for (auto a: drinks) {
        cout << a << endl;
      }
    }
    void AddAppetizer(string app) {
      appetizers.push_back(app);
    }
    void PrintAppetizers() {
      for (auto a: appetizers) {
        cout << a << endl;
      }
    }
    void AddMainCourse(string mc) {
      main_courses.push_back(mc);
    }
    void PrintMainCourses() {
      for (auto a: main_courses) {
        cout << a << endl;
      }
    }
    void AddDessert(string dessert) {
      desserts.push_back(dessert);
    }
    void PrintDesserts() {
      for (auto a: desserts) {
        cout << a << endl;
      }
    }
```

```
  private:
    vector<string> drinks;
    vector<string> appetizers;
    vector<string> main_courses;
    vector<string> desserts;
};

//add class definitions above this line

int main() {

  //add code below this line

  Meal dinner;
  dinner.AddDrink("water");
  dinner.PrintDrinks();
  dinner.AddAppetizer("bruschetta");
  dinner.PrintAppetizers();
  dinner.AddMainCourse("roast chicken");
  dinner.PrintMainCourses();
  dinner.AddDessert("chocolate cake");
  dinner.PrintDesserts();

  //add code above this line

  return 0;

}
```

Printing the Meal

Format the Entire Meal

Currently, you should have the following code the text editor:

```
#include <iostream>
#include <vector>
using namespace std;

//add class definitions below this line

class Meal {
  public:
    void AddDrink(string drink) {
      drinks.push_back(drink);
    }
    void PrintDrinks() {
      for (auto a: drinks) {
        cout << a << endl;
      }
    }
    void AddAppetizer(string app) {
      appetizers.push_back(app);
    }
    void PrintAppetizers() {
      for (auto a: appetizers) {
        cout << a << endl;
      }
    }
    void AddMainCourse(string mc) {
      main_courses.push_back(mc);
    }
    void PrintMainCourses() {
      for (auto a: main_courses) {
        cout << a << endl;
      }
    }
    void AddDessert(string dessert) {
      desserts.push_back(dessert);
    }
    void PrintDesserts() {
```

```
      for (auto a: desserts) {
        cout << a << endl;
      }
    }

  private:
    vector<string> drinks;
    vector<string> appetizers;
    vector<string> main_courses;
    vector<string> desserts;
};

//add class definitions above this line

int main() {

  //add code below this line

  Meal dinner;
  dinner.AddDrink("water");
  dinner.PrintDrinks();
  dinner.AddAppetizer("bruschetta");
  dinner.PrintAppetizers();
  dinner.AddMainCourse("roast chicken");
  dinner.PrintMainCourses();
  dinner.AddDessert("chocolate cake");
  dinner.PrintDesserts();

  //add code above this line

  return 0;

}
```

What you want to do next is to print **all** of the Meal attributes that you have in a format that is concise, clear, and makes sense. The ideal output should also take into consideration how many items there are. Here are some sample output:

0 Item in the Category

```
Drinks: None
```

1 Item in the Category

```
Drinks: water
```

2 Items in the Category

```
Drinks: water and lemonade
```

3 Items in the Category

```
Drinks: water, lemonade, and tea
```

Modify the Class Functions

In order to print the desired results, you'll have to modify the class functions. In particular, you'll want to include multiple conditionals to select for the printing format that you want. Note that currently, an enhanced `for` loop is used to iterate the vectors. The first three sample output is more straightforward because you can just use the `size` function to check the vector's size and then use the `at` function to print either `0`, `1`, or `2` items like below:

```
void PrintDrinks() {
  if (drinks.size() == 0) {
    cout << "Drinks: None" << endl;
  }
  else if (drinks.size() == 1) {
    cout << "Drinks: " << drinks.at(0) << endl;
  }
  else if (drinks.size() == 2) {
    cout << "Drinks: " << drinks.at(0) << " and " <<
        drinks.at(1) << endl;
  }
}
```

However, how will you select for categories that have 3 or more items? See if you can try doing so on your own. Run your code a couple of times using `PrintDrinks` after adding different amounts of elements using `AddDrink`.

You can compare your solution to a sample one below:

▼ **Sample Code**

```
void PrintDrinks() { //class definition
  if (drinks.size() == 0) {
    cout << "Drinks: None" << endl;
  }
  else if (drinks.size() == 1) {
    cout << "Drinks: " << drinks.at(0) << endl;
  }
  else if (drinks.size() == 2) {
    cout << "Drinks: " << drinks.at(0) << " and " <<
        drinks.at(1) << endl;
  }
  else {
    cout << "Drinks: ";
    for (int i = 0; i < drinks.size() - 1; i++) {
      cout << drinks.at(i) << ", ";
    }
    cout << "and " << drinks.at(drinks.size() - 1) << endl;
  }
}
```

```
  Meal dinner;
  dinner.PrintDrinks();
  dinner.AddDrink("water");
  dinner.PrintDrinks();
  dinner.AddDrink("lemonade");
  dinner.PrintDrinks();
  dinner.AddDrink("tea");
  dinner.PrintDrinks();
  dinner.AddDrink("coffee");
  dinner.PrintDrinks();
```

Print the Entire Meal

Once you've completed modifying one class function. You can modify the rest as specified. To print the entire meal, create a class function called `PrintMeal` that incorporates all of the other `Print` functions. Once you've done that, test your code with the following statements in `main`:

```
Meal dinner;
dinner.AddDrink("Pepsi");
dinner.AddDrink("Sprite");
dinner.AddAppetizer("egg rolls");
dinner.AddAppetizer("pot stickers");
dinner.AddAppetizer("buffalo wings");
dinner.AddMainCourse("smoked salmon");
dinner.PrintMeal();
```

Expected Output

```
Drink(s): Pepsi and Sprite
Appetizer(s): egg rolls, pot stickers, and buffalo wings
Main Course(s): smoked salmon
Dessert(s): None
```

▼ Sample Code

```
#include <iostream>
#include <vector>
using namespace std;

//add class definitions below this line

class Meal {
  public:
    void AddDrink(string drink) {
      drinks.push_back(drink);
    }
    void PrintDrinks() {
      if (drinks.size() == 0) {
        cout << "Drink(s): None" << endl;
      }
      else if (drinks.size() == 1) {
        cout << "Drink(s): " << drinks.at(0) << endl;
      }
      else if (drinks.size() == 2) {
        cout << "Drink(s): " << drinks.at(0) << " and " <<
        drinks.at(1) << endl;
      }
      else {
        cout << "Drink(s): ";
        for (int i = 0; i < drinks.size() - 1; i++) {
          cout << drinks.at(i) << ", ";
        }
```

```
    cout << "and " << drinks.at(drinks.size() - 1) << endl;
  }
}
void AddAppetizer(string app) {
  appetizers.push_back(app);
}
void PrintAppetizers() {
  if (appetizers.size() == 0) {
    cout << "Appetizers(s): None" << endl;
  }
  else if (appetizers.size() == 1) {
    cout << "Appetizers(s): " << appetizers.at(0) << endl;
  }
  else if (appetizers.size() == 2) {
    cout << "Appetizers(s): " << appetizers.at(0) << " and "
    << appetizers.at(1) << endl;
  }
  else {
    cout << "Appetizers(s): ";
    for (int i = 0; i < appetizers.size() - 1; i++) {
      cout << appetizers.at(i) << ", ";
    }
    cout << "and " << appetizers.at(appetizers.size() - 1)
    << endl;
  }
}
void AddMainCourse(string mc) {
  main_courses.push_back(mc);
}
void PrintMainCourses() {
  if (main_courses.size() == 0) {
    cout << "Main Course(s): None" << endl;
  }
  else if (main_courses.size() == 1) {
    cout << "Main Course(s): " << main_courses.at(0) <<
    endl;
  }
  else if (main_courses.size() == 2) {
    cout << "Main Course(s): " << main_courses.at(0) << "
    and " << main_courses.at(1) << endl;
  }
  else {
    cout << "Main Course(s): ";
    for (int i = 0; i < main_courses.size() - 1; i++) {
      cout << main_courses.at(i) << ", ";
    }
    cout << "and " << main_courses.at(main_courses.size() -
    1) << endl;
  }
}
```

```
    void AddDessert(string dessert) {
      desserts.push_back(dessert);
    }
    void PrintDesserts() {
      if (desserts.size() == 0) {
        cout << "Dessert(s): None" << endl;
      }
      else if (desserts.size() == 1) {
        cout << "Dessert(s): " << desserts.at(0) << endl;
      }
      else if (desserts.size() == 2) {
        cout << "Dessert(s): " << desserts.at(0) << " and " <<
        desserts.at(1) << endl;
      }
      else {
        cout << "Dessert(s): ";
        for (int i = 0; i < desserts.size() - 1; i++) {
          cout << desserts.at(i) << ", ";
        }
        cout << "and " << desserts.at(desserts.size() - 1) <<
        endl;
      }
    }
    void PrintMeal() {
      PrintDrinks();
      PrintAppetizers();
      PrintMainCourses();
      PrintDesserts();
    }

  private:
    vector<string> drinks;
    vector<string> appetizers;
    vector<string> main_courses;
    vector<string> desserts;
};

//add class definitions above this line

int main() {

  //add code below this line

Meal dinner;
dinner.AddDrink("Pepsi");
dinner.AddDrink("Sprite");
dinner.AddAppetizer("egg rolls");
dinner.AddAppetizer("pot stickers");
dinner.AddAppetizer("buffalo wings");
```

```
dinner.AddMainCourse("smoked salmon");
dinner.PrintMeal();

  //add code above this line

  return 0;

}
```