Chapter 9 introduces the technique of recursive programming. As you have seen, recursive programming involves spotting smaller occurrences of a problem within the problem itself. This presentation gives an additional example, which is not in the book.

To start the example, think about your favorite family car.

Imagine that the car is controlled by a radio signal from a computer.
A Car Class

To start the example, think about your favorite family car.
Imagine that the car is controlled by a radio signal from a computer.
The radio signals are generated by activating member functions of a Car object.

```cpp
class Car
{
public:
    Car(int car_number);
    void move();
    void turn_around();
    bool is_blocked;
private:
    // We don't need to know the private fields!
};
```

member Functions for the Car Class

The Constructor

When we declare a Car and activate the constructor, the computer makes a radio link with a car that has a particular number.

```cpp
int main()
{
    Car racer(7);
    ...
}
```

The turn_around Function

When we activate turn_around, the computer signals the car to turn 180 degrees.

```cpp
int main()
{
    Car racer(7);
    racer.turn_around();
    ...
}
```

The move Function

When we activate move, the computer signals the car to move forward one foot.

```cpp
int main()
{
    Car racer(7);
    racer.turn_around();
    racer.move();
    ...
}
```

The move Function

When we activate move, the computer signals the car to move forward one foot.

```cpp
int main()
{
    Car racer(7);
    racer.turn_around();
    racer.move();
    ...
}
```
The is_blocked( ) Function

```c
int main( )
{
    Car racer(7);
    racer.turn_around( );
    racer.move( );
    if (racer.is_blocked( ))
        cout << "Cannot move!";
    ...
}
```

The is_blocked member function detects barriers.

Your Mission

- Write a function which will move a Car forward until it reaches a barrier...

Your Mission

- Write a function which will move a Car forward until it reaches a barrier...

Your Mission

- Write a function which will move a Car forward until it reaches a barrier...
  - then the car is turned around...
  - and returned to its original location, facing the opposite way.
Your Mission

- Write a function which will move a Car forward until it reaches a barrier...
- ...then the car is turned around...
- ...and returned to its original location, facing the opposite way.

Pseudocode for ricochet

```c
void ricochet(Car& moving_car);
```

1. if moving_car.is_blocked(), then the car is already at the barrier. In this case, just turn the car around.

2. Otherwise, the car has not yet reached the barrier, so start with:
   ```c
   moving_car.move();
   ```

This makes the problem a bit smaller. For example, if the car started 100 feet from the barrier... then after activating `move` once, the distance is only 99 feet.
Pseudocode for ricochet

```c
void ricochet(Car& moving_car);
```

1. If `moving_car.is_blocked()` (false), then the car is already at the barrier. In this case, just turn the car around.
2. Otherwise, the car has not yet reached the barrier, so start with:

   ```c
   moving_car.move();
   ricochet(moving_car);
   ```

We now have a smaller version of the same problem that we started with.

Make a recursive call to solve the smaller problem.

The recursive call will solve the smaller problem.

The recursive call will solve the smaller problem.

The recursive call will solve the smaller problem.
Pseudocode for ricochet

```cpp
void ricochet(Car& moving_car) {
    if moving_car.is_blocked() { // The car is already at the barrier.
        // Just turn the car around.
    } else { // The car has not yet reached the barrier, so
        moving_car.move();
        ricochet(moving_car); // The recursive call will solve the smaller problem.
        . . .
    }
}
```

99 ft.
Pseudocode for ricochet

```c
void ricochet(Car& moving_car);
```

1. if moving_car.is_blocked(), then the car is already at the barrier. In this case, just turn the car around.
2. Otherwise, the car has not yet reached the barrier, so start with:

```
moving_car.move();
ricochet(moving_car);
```

---

**What is the last step that's needed to return to our original location?**

This recursive function follows a common pattern that you should recognize.

---

When the problem is simple, solve it with no recursive call. This is the **base case**.

---

When the problem is more complex, start by doing work to create a smaller version of the same problem...

---

...use a **recursive call** to completely solve the smaller problem...
Pseudocode for ricochet

```
if moving_car.is_blocked(), then the car is already at the barrier. In this case, just turn the car around.

Otherwise, the car has not yet reached the barrier, so start with:

moving_car.move();
ricochet(moving_car);
moving_car.move();

...and finally do any work that’s needed to complete the solution of the original problem.
```

Implementation of ricochet

```
void ricochet(Car& moving_car)
{
   if (moving_car.is_blocked())
      moving_car.turn_around(); // Base case
   else
      { // Recursive pattern
         moving_car.move();
         ricochet(moving_car);
         moving_car.move();
      }
}
```

An Exercise

```
Can you write ricochet as a new member function of the Car class, instead of a separate function?

void Car::ricochet()
{
   ... 
}
```

An Exercise

```
One solution:

void Car::ricochet()
{
   if (is_blocked())
      turn_around(); // Base case
   else
      { // Recursive pattern
         move();
         ricochet();
         move();
      }
}
```

Presentation copyright 1997 Addison Wesley Longman, For use with Data Structures and Other Objects Using C++ by Michael Main and Walter Savitch.

Some artwork in the presentation is used with permission from Presentation Task Force (copyright New Vision Technologies, Inc.), Corel Gallery Clipart Catalog (copyright Corel Corporation, 3G Graphics Inc., Archive Arts, Camensa Software, Image Club Graphics Inc., One Mile Up Inc., TechPics Studios, Tidels Graphics Inc). Students and instructors who use Data Structures and Other Objects Using C++ are welcome to use this presentation however they see fit, so long as this copyright notice remains intact.

THE END