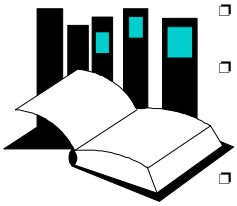
Recursive Thinking

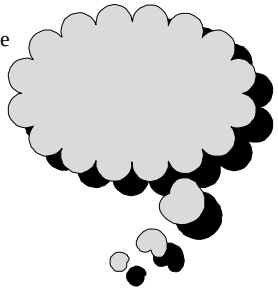


- Chapter 9 introduces the technique of recursive programming.
- As you have seen, recursive programming involves spotting smaller occurrences of a problem within the problem itself.
- This presentation gives an additional example, which is not in the book.

Data Structures and Other Objects Using C++

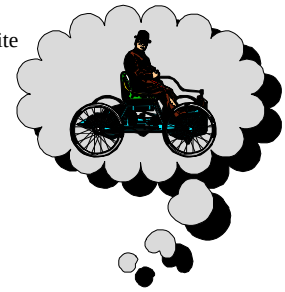
A Car Object

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



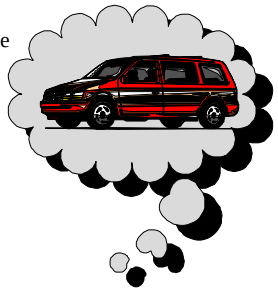
A Car Object

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



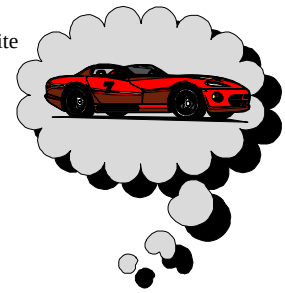
A Car Object

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



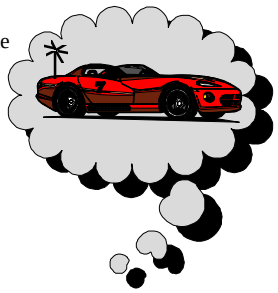
A Car Object

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



A Car Object

- To start the example, think about your favorite family car
- Imagine that the car is controlled by a radio signal from a computer



7

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

```
class Car
{
public:
        . . .
};
```

8

Member Functions for the Car Class

```
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! }
        . . .
};
```

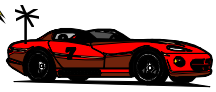
9

The Constructor

```
int main( )
{
   Car racer(7);

   . . .
```

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



10

The turn_around Function

```
int main( )
{
   Car racer(7);

   racer.turn_around( );
   . . .
```

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

11

The move Function

```
int main( )
{
   Car racer(7);

   racer.turn_around( );
   racer.move( );
   . . .
```

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

12

The move Function

```
int main( )
{
   Car racer(7);

   racer.turn_around( );
   racer.move( );
   . . .
```

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

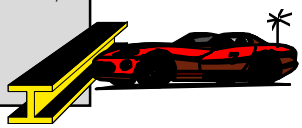
13

The is_blocked( ) Function

```
int main( )
{
   Car racer(7);

   racer.turn_around( );
   racer.move( );
   if (racer.is_blocked( ) )
     cout << "Cannot move!";
   . . .
```

The is_blocked member function detects barriers.



14

Your Mission

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

15

Your Mission

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

16

Your Mission

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

17

Your Mission

- Write a function which will move a Car forward until it reaches a barrier...
- ...then the car is turned around...

18

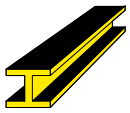
Your Mission

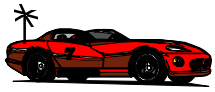
- Write a function which will move a Car forward until it reaches a barrier...
- ...then the car is turned around...
- ...and returned to its original location, facing the opposite way.

19

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.





20

Your Mission

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

- 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.

21

Pseudocode for ricochet

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

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

22

Pseudocode for ricochet

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

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

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

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

23

Pseudocode for ricochet

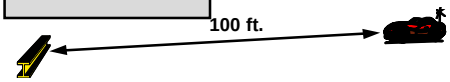
```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

```
moving_car.move(
. . .
```

This makes the problem a bit **smaller**. For example, if the car started 100 feet from the barrier...



24

Pseudocode for ricochet

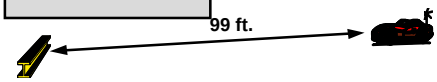
```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

```
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

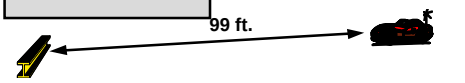
```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

```
moving_car.move(
. . .
```

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



Pseudocode for ricochet

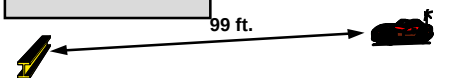
```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

Make a recursive call to solve the smaller problem.



Pseudocode for ricochet

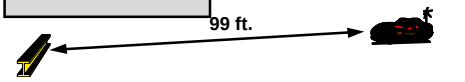
```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

The recursive call will solve the smaller problem.



Pseudocode for ricochet

```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

The recursive call will solve the smaller problem.

Pseudocode for ricochet

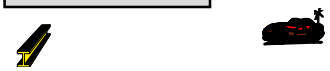
```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

The recursive call will solve the smaller problem.



Pseudocode for ricochet

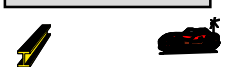
```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

The recursive call will solve the smaller problem.



31

Pseudocode for ricochet

```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

The recursive call will solve the smaller problem.

32

Pseudocode for ricochet

```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

The recursive call will solve the smaller problem.

33

Pseudocode for ricochet

```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

The recursive call will solve the smaller problem.

34

Pseudocode for ricochet

```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

The recursive call will solve the smaller problem.

35

Pseudocode for ricochet

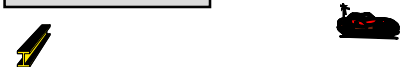
```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

The recursive call will solve the smaller problem.



36

Pseudocode for ricochet

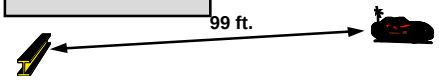
```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

The recursive call will solve the smaller problem.



Pseudocode for ricochet

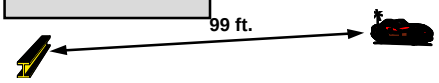
```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

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



Pseudocode for ricochet

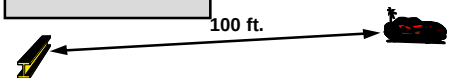
```
void ricochet(Car& movin
```

❶ if moving_car.is_blo
the barrier. In this ca

❷ Otherwise, the car ha
start with:

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

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



Pseudocode for ricochet

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

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

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

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

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

Pseudocode for ricochet

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

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

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

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

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

Pseudocode for ricochet

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

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

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

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

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

Pseudocode for ricochet

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

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

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

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

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

Pseudocode for ricochet

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

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

❷ 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

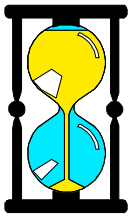
```
void ricochet(Car& moving_car)
{
   if (moving_car.is_blocked( ))
      moving_car.turn_around( ); // Base case
   else
   {     // Recursive pattern
         moving_car.move( );
         ricochet(moving_car);
         moving_car.move( );
    }
}
```

Look for this pattern in the other examples of Chapter 9.

An Exercise

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

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



You have 2 minutes to write the implementation.

An Exercise

One solution:

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


THE END