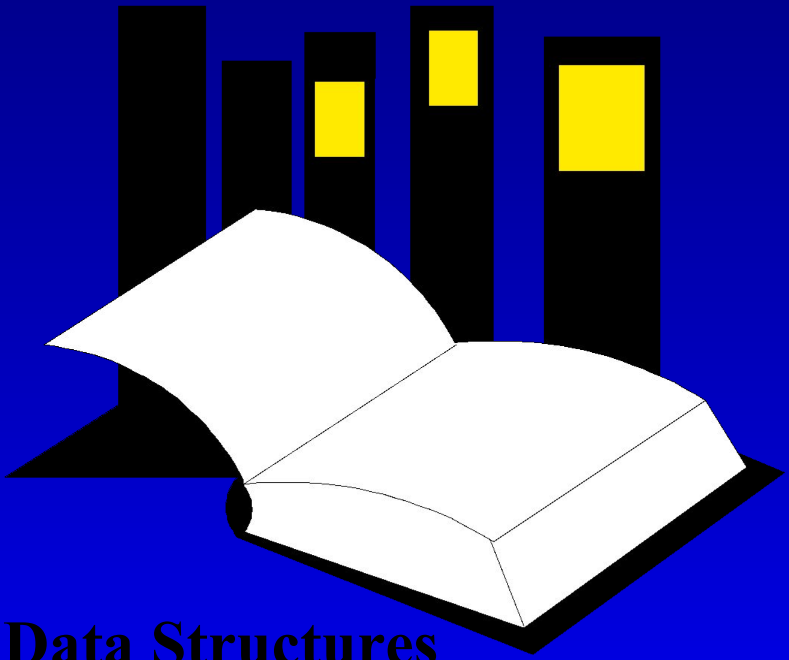
Object Oriented Programming

OOP

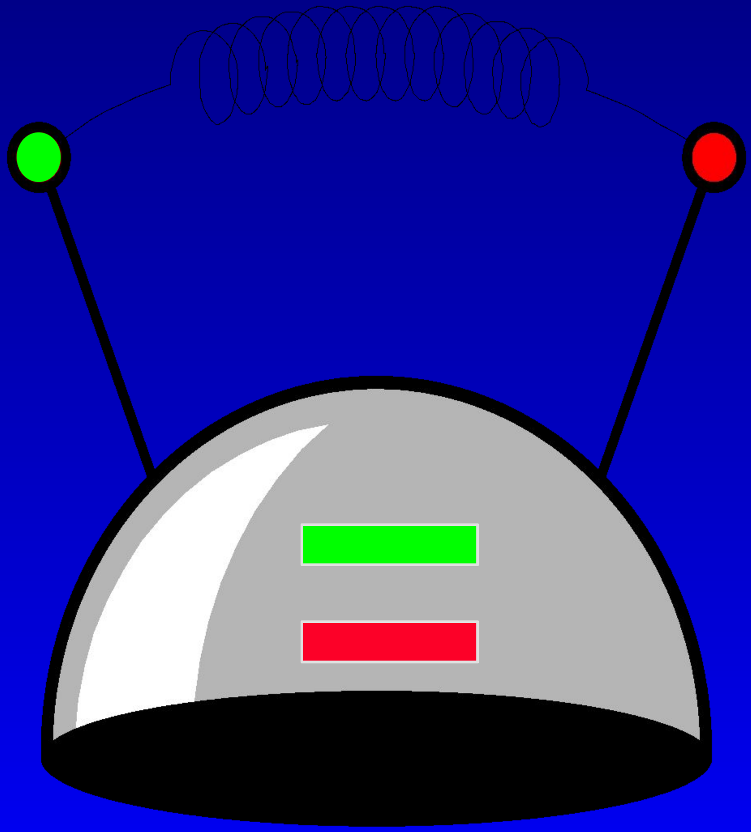


Data Structures and Other Objects Using C++

- Chapter 2 introduces Object Oriented Programming.
- OOP is a relatively new approach to programming which supports the creation of new data types and operations to manipulate those types.
- This presentation introduces OOP.

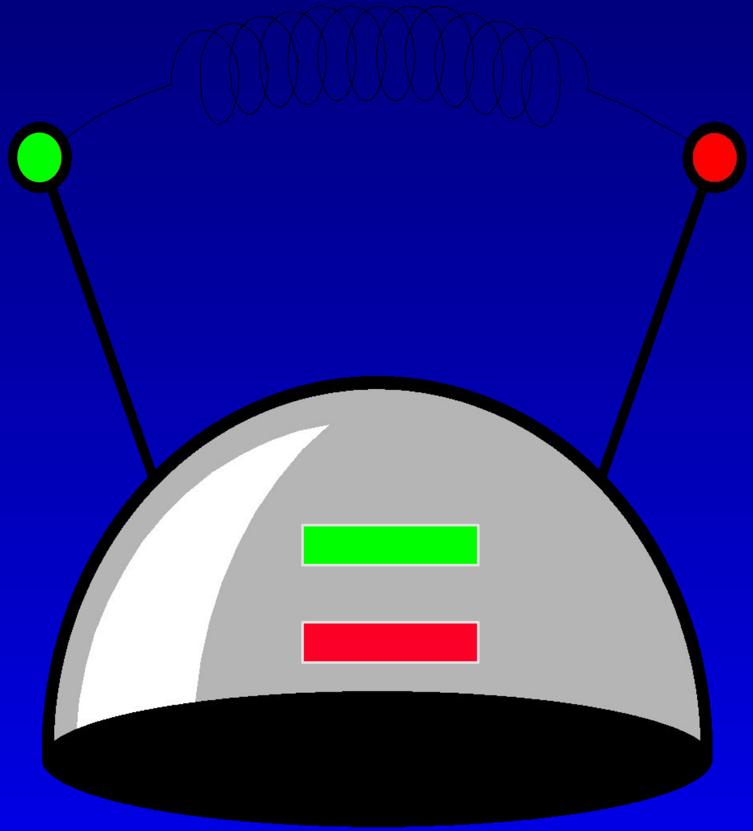
What is this Object ?

- There is no real answer to the question, but we’ll call it a “thinking cap”.
- The plan is to describe a thinking cap by telling you what actions can be done to it.

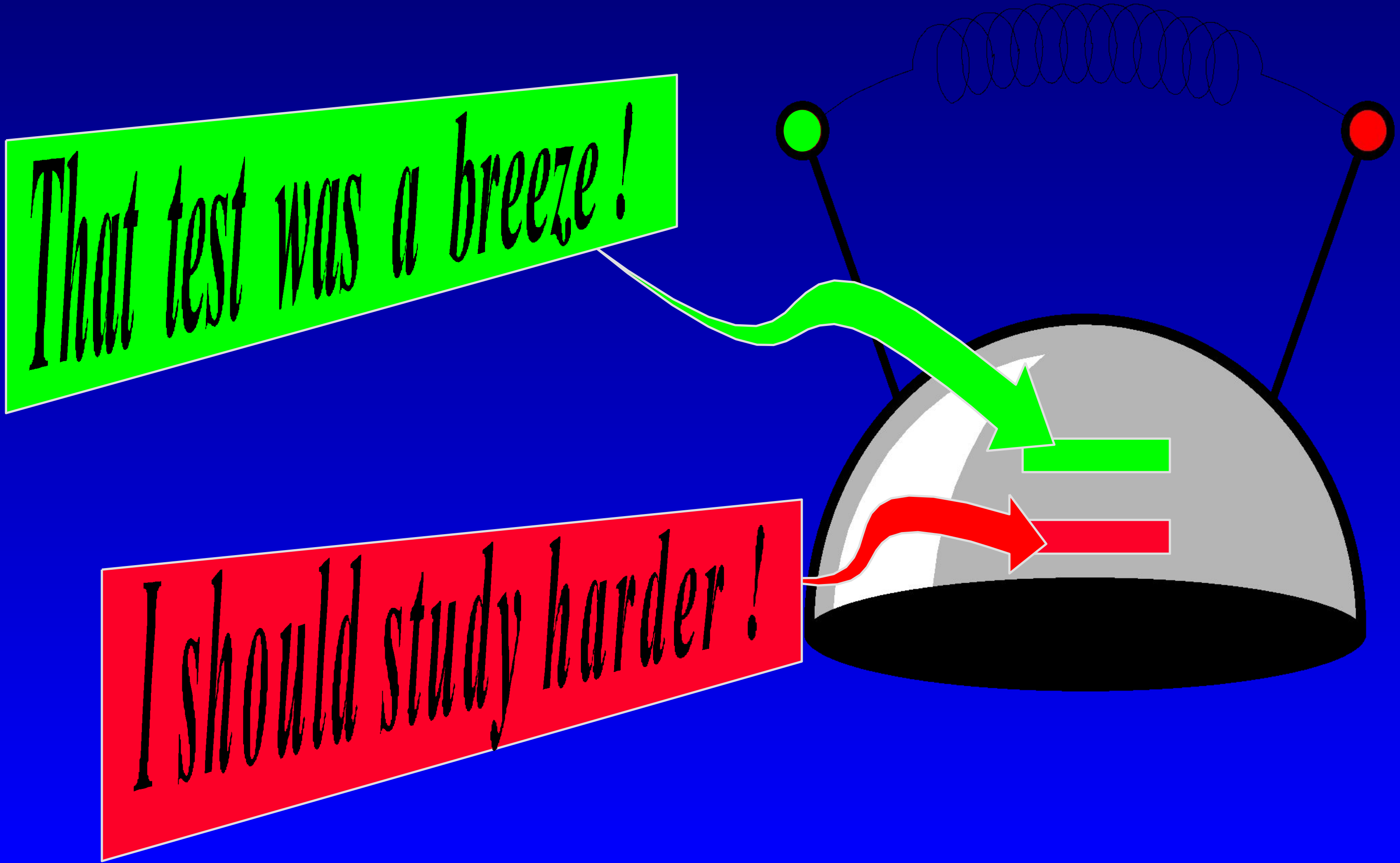


Using the Object's Slots

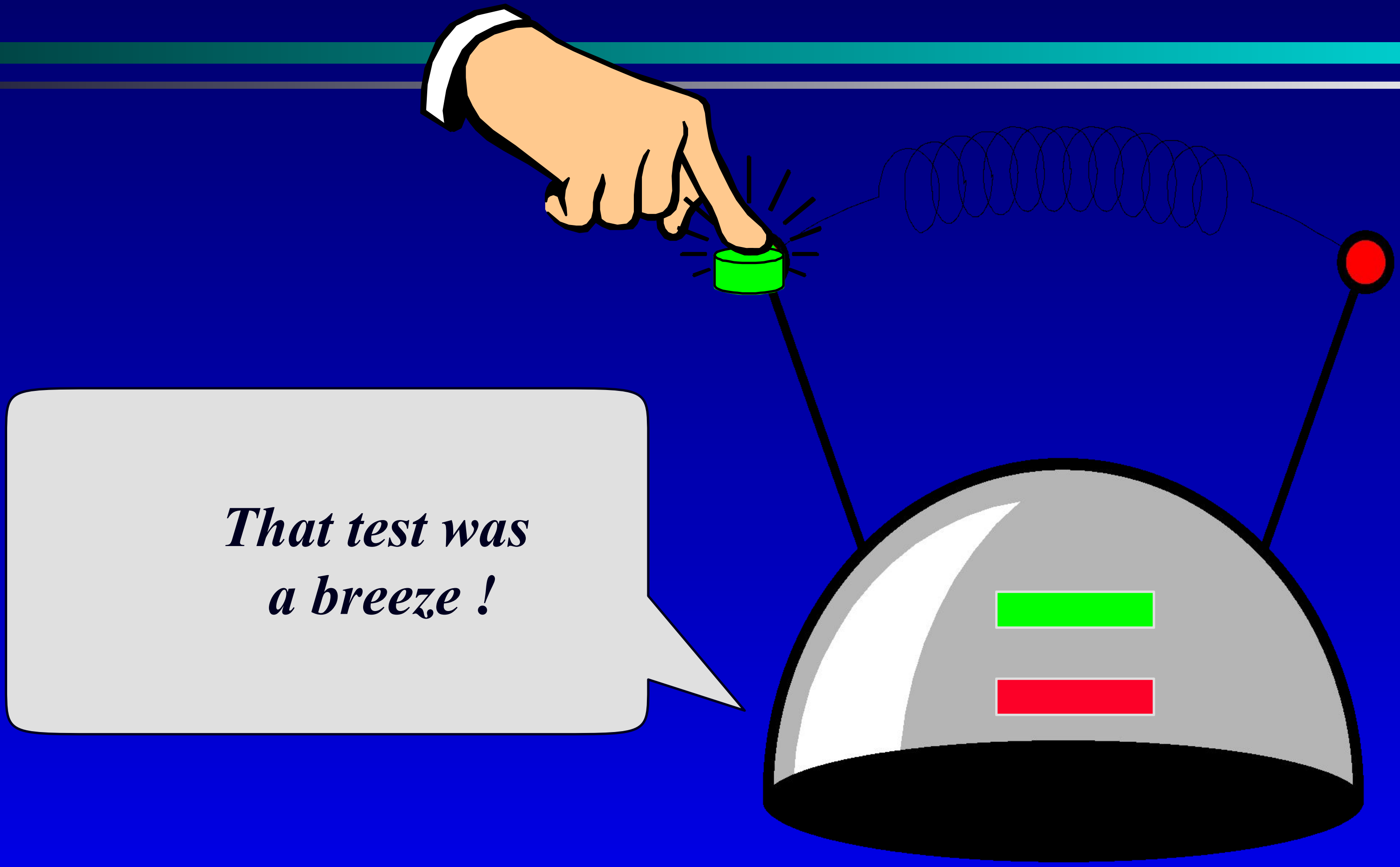
- You may put a piece of paper in each of the two slots (green and red), with a sentence written on each.
- You may push the green button and the thinking cap will speak the sentence from the green slot's paper.
- And same for the red button.



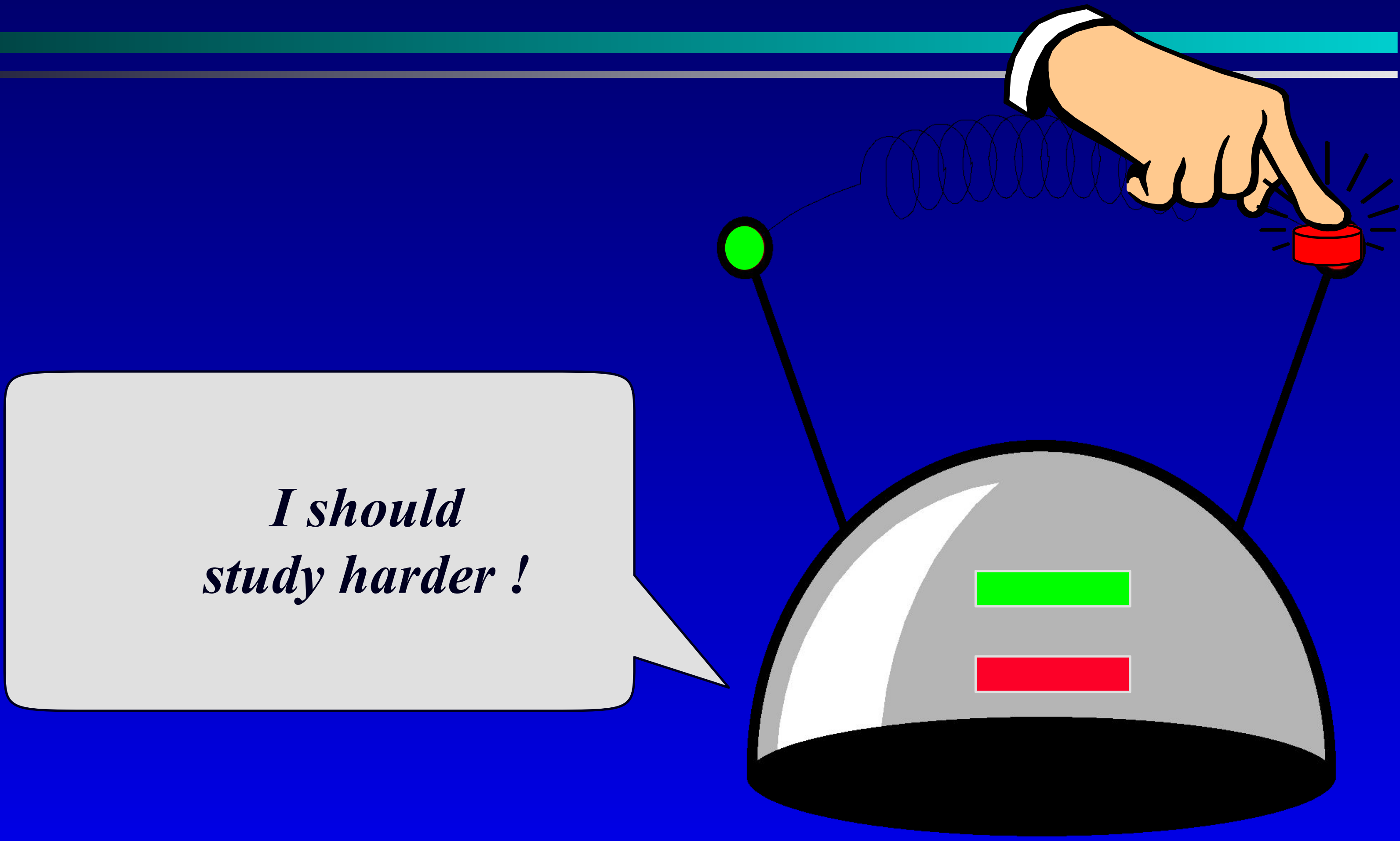
Example



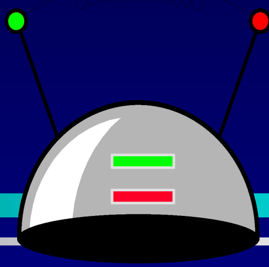
Example



Example



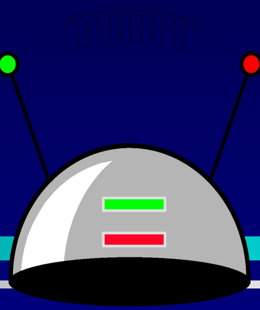
Thinking Cap Implementation



- We can implement the thinking cap using a data type called a class.

```
class thinking_cap
{

            . . .

 };
```

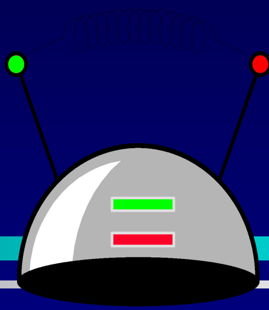
Thinking Cap Implementation



- The class will have two components called green_string and red_string. These compnents are strings which hold the information that is placed in the two slots.
- Using a class permits two new features . . .

```
class thinking_cap
{
    . . .
    char green_string[50];
    char red_string[50];

};
```

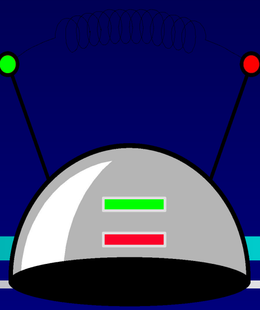
Thinking Cap Implementation



- The two components will be private member variables. This ensures that nobody can directly access this information. The only access is through functions that we provide for the class.

```
class thinking_cap
{
    . . .
private:
    char green_string[50];
    char red_string[50];
};
```

Thinking Cap Implementation

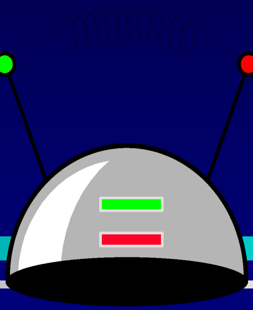


- In a class, the functions which manipulate the class are also listed.

```
class thinking_cap
{
public:
      . . .
private:
      char green_string[50];
      char red_string[50];
};
```

Prototypes for the thinking cap functions go here, after the word <u>public:</u>

Thinking Cap Implementation

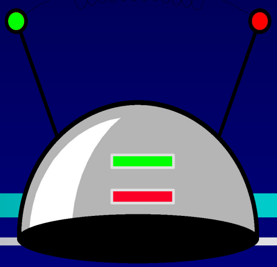


- In a class, the functions which manipulate the class are also listed.

```
class thinking_cap
{
public:
    . . .
private:
    char green_string[50];
    char red_string[50];
};
```

Prototypes for the thinking cap <u>member functions</u> go here

Thinking Cap Implementation

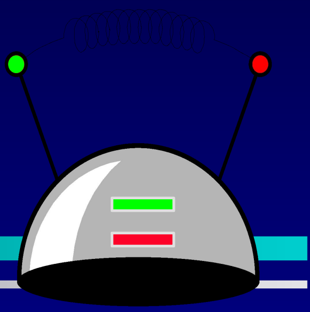


Our thinking cap has at least three member functions:

```
class thinking_cap
{
public:
    void slots(char new_green[ ], char new_red[ ]);
    void push_green( ) const;
    void push_red( ) const;
private:
    char green_string[50];
    char red_string[50];
};
```

Function bodies
will be elsewhere.

Thinking Cap Implementation



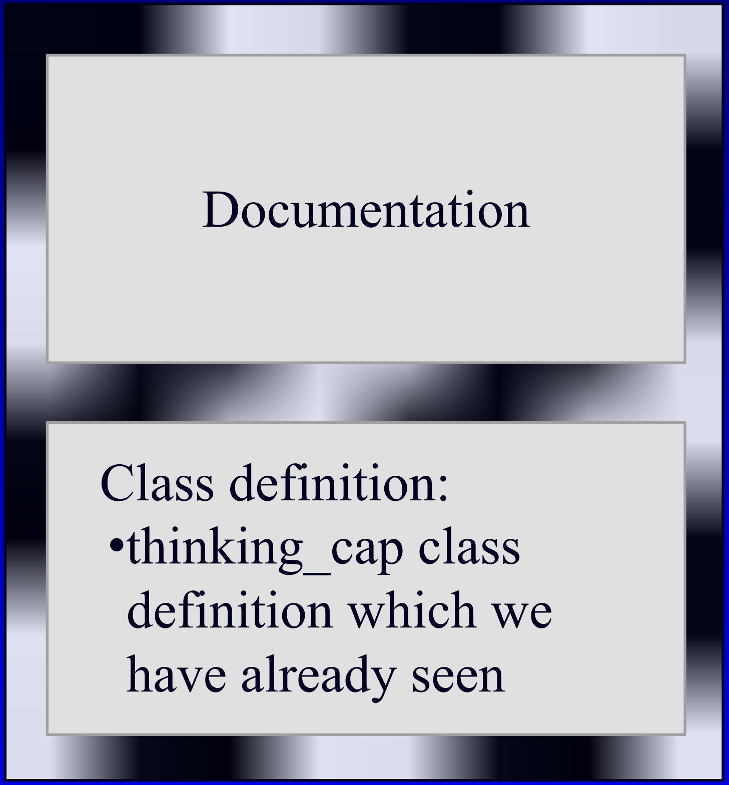
The keyword **const** appears after two prototypes:

```
class thinking_cap
{
public:
    void slots(char new_green[ ], char new            );
    void push_green( ) const;
    void push_red( ) const;
private:
    char green_string[50];
    char red_string[50];
};
```

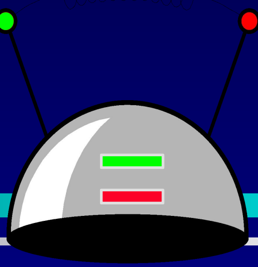
This means that these functions will not change the data stored in a thinking_cap.

Files for the Thinking Cap

- The thinking_cap class definition, which we have just seen, is placed with documentation in a file called thinker.h, outlined here.
- The implementations of the three member functions will be placed in a separate file called thinker.cxx, which we will examine in a few minutes.



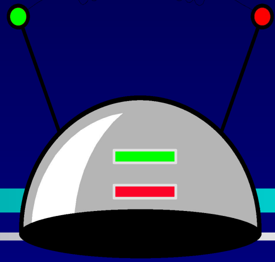
Using the Thinking Cap



- A program that wants to use the thinking cap must **include** the thinker header file (along with its other header inclusions).

```
#include <iostream.h>
#include <stdlib.h>
#include "thinker.h"

...
```

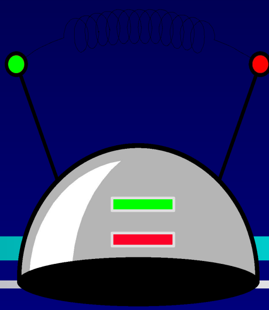
Using the Thinking Cap



- Just for fun, the example program will declare two thinking_cap variables named student and fan.

```
#include <iostream.h>
#include <stdlib.h>
#include "thinker.h"

int main( )
{
    thinking_cap student:
    thinking_cap fan;
```

Using the Thinking Cap



- Just for fun, the example program will declare two thinking_cap objects named student and fan.

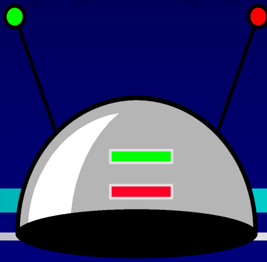
```
#include <iostream.h>
#include <stdlib.h>
#include "thinker.h"

int main( )
{
    thinking_cap student;
    thinking_cap fan;
```

Using the Thinking Cap

- The program starts by calling the slots member function for student.

```
#include <iostream.h>
#include <stdlib.h>
#include "thinker.h"

int main( )
{
    thinking_cap student;
    thinking_cap fan;
    student.slots( "Hello",  "Goodbye");
```

Using the Thinking Cap



- The program starts by activating the slots member function for student.

```
#include <iostream.h>
#include <stdlib.h>
#include "thinker.h"

int main( )
{
    thinking_cap student:
    thinking_cap fan;

    student.slots( "Hello",  "Goodbye");
```

Using the Thinking Cap

- The member function activation consists of four parts, starting with the object name.

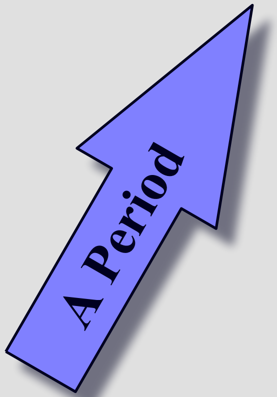
```
int main( )
 {
   thinking_cap student;
   thinking_cap fan;

   student.slots( "Hello",  "Goodbye");
```

Name of the object

Using the Thinking Cap

- The instance name is followed by a period.

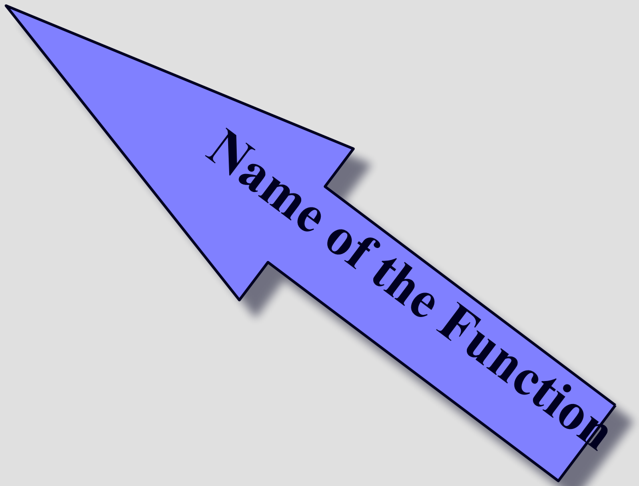
```
int main( )
{
    thinking_cap student;
    thinking_cap fan;

    student.slots( "Hello",  "Goodbye");
```



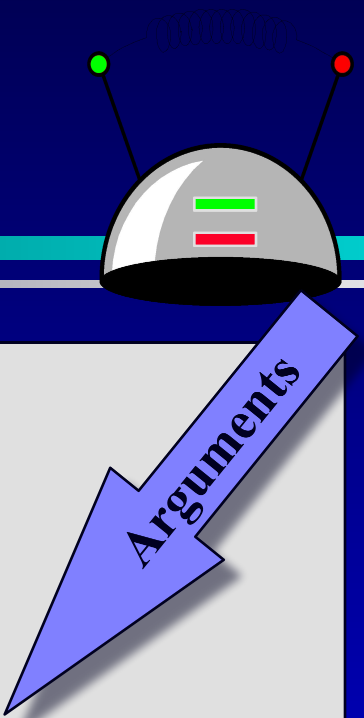
Using the Thinking Cap

- After the period is the name of the member function that you are activating.

```
int main( ) {
    thinking_cap student;
    thinking_cap fan;

    student.slots( "Hello",  "Goodbye");
```



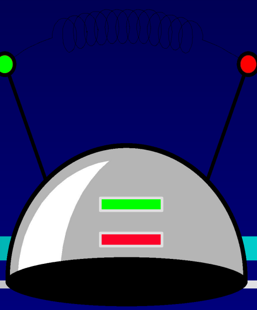
Using the Thinking Cap



- Finally, the arguments for the member function. In this example the first argument (new_green) is "Hello" and the second argument (new_red) is "Goodbye".

```
#include "thinker.h"

int main( ) {
    thinking_cap student;
    thinking_cap fan;

    student.slots( "Hello",  "Goodbye");
```

A Quiz

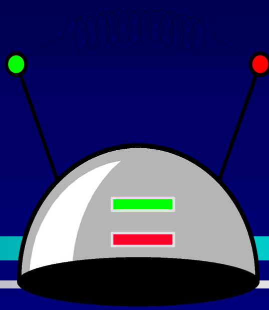


How would you activate student's push_green member function ?

What would be the output of student's push_green member function at this point in the program ?

```
int main( )
{
    thinking_cap student;
    thinking_cap fan;

    student.slots( "Hello",  "Goodbye");
```

A Quiz



Notice that the **push_green** member function has no arguments.

At this point, activating **student.push_green** will print the string **Hello**.

```
int main( ) {
    thinking_cap student;
    thinking_cap fan;

    student.slots( "Hello",  "Goodbye");

    student.push_green( );
```

A Quiz

```
int main( )
{
    thinking_cap student;
    thinking_cap fan;
    student.slots( "Hello",  "Goodbye");
    fan.slots( "Go Cougars!", "Boo!");
    student.push_green( );
    fan.push_green( );
    student.push_red( );
    . . .
```

Trace through this program, and tell me the complete output.

A Quiz

```
int main( )
 {
   thinking_cap student;
   thinking_cap fan;
   student.slots( "Hello",  "Goodbye");
   fan.slots( "Go Cougars!", "Boo!");
   student.push_green( );
   fan.push_green( );
   student.push_red( );
   . . .
```

Hello

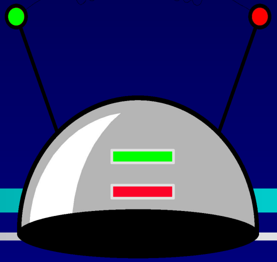
Go Cougars!

Goodbye

What you know about Objects

- Class = Data + Member Functions.
- You know how to define a new class type, and place the definition in a header file.
- You know how to use the header file in a program which declares instances of the class type.
- You know how to activate member functions.
- But you still need to learn how to write the bodies of a class's member functions.

Thinking Cap Implementation

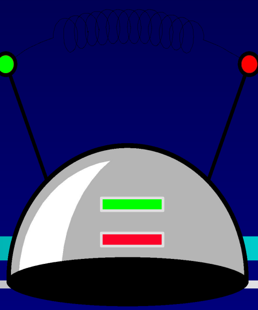


Remember that the member function's bodies generally appear in a separate .cxx file.

```
class thinking_cap
{
public:
    void slots(char new_green[ ], char new_red[ ]);
    void push_green( );
    void push_red( );
private:
    char green_string[50];
    char red_string[50];
};
```

Function bodies will be in .cxx file.

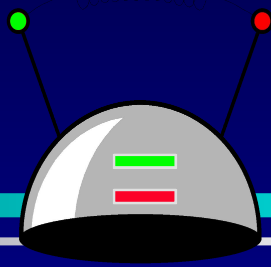
Thinking Cap Implementation



We will look at the body of slots, which must copy its two arguments to the two private member variables.

```
class thinking_cap
{
public:
    void slots(char new_green[ ], char new_red[ ]);
    void push_green( );
    void push_red( );
private:
    char green_string[50];
    char red_string[50];
};
```

Thinking Cap Implementation



For the most part, the function's body is no different than any other function body.

```
void thinking_cap::slots(char new_green[ ], char new_red[ ])
{
    assert(strlen(new_green) < 50);
    assert(strlen(new_red) < 50);
    strcpy(green_string,  new_green);
    strcpy(red_string, new_red);
}
```

But there are two special features about a member function's body . . .

Thinking Cap Implementation

- In the heading, the function's name is preceded by the class name and :: - otherwise C++ won't realize this is a class’s member function.

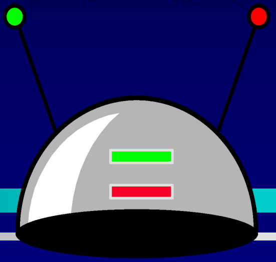
```
void thinking_cap::slots(char new_green[ ], char new_red[ ])
{
    assert(strlen(new_green) < 50);
    assert(strlen(new_red) < 50);
    strcpy(green_string,  new_green);
    strcpy(red_string, new_red);
}
```

Thinking Cap Implementation

- Within the body of the function, the class's member variables and other member functions may all be accessed.

```
void thinking_cap::slots(char new_green[ ], char new_red[ ])
{
    assert(strlen(new_green) < 50);
    assert(strlen(new_red) < 50);
    strcpy(green_string,  new_green);
    strcpy(red_string, new_red);
}
```

Thinking Cap Implementation



- Within the body of the function, the class's member variables and other member functions may all be accessed.

```
void thinking_cap::slots(char new_
{
    assert(strlen(new_green) < 50)
    assert(strlen(new_red) < 50);
    strcpy(green_string,  new_gre
    strcpy(red_string, new_red);
}
```

But, whose member variables are these? Are they

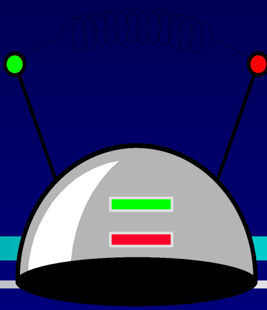
student.green_string

student.red_string

fan.green_string

fan.red_string

Thinking Cap Implementation



- Within the body of the function, the class's member variables and other member functions may all be accessed.

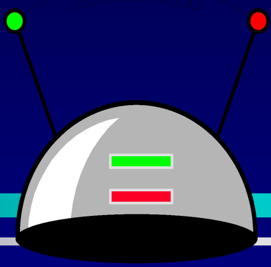
```
void thinking_cap::slots(char new_
{
    assert(strlen(new_green) < 50)
    assert(strlen(new_red) < 50);
    strcpy(green_string,  new_gre
    strcpy(red_string, new_red);
}
```

If we activate student.slots:

student.green_string

student.red_string

Thinking Cap Implementation



- Within the body of the function, the class's member variables and other member functions may all be accessed.

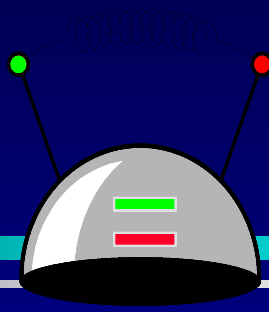
```
void thinking_cap::slots(char new_
{
    assert(strlen(new_green) < 50)
    assert(strlen(new_red) < 50);
    strcpy(green_string,  new_gre
    strcpy(red_string, new_red);
}
```

If we activate fan.slots:

fan.green_string

fan.red_string

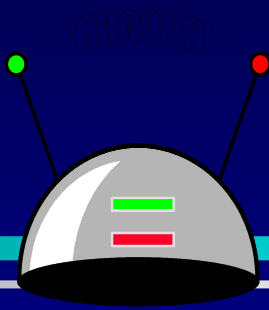
Thinking Cap Implementation



Here is the implementation of the push_green member function, which prints the green message:

```
void thinking_cap::push_green
{

    cout << green_string << endl;

}
```

Thinking Cap Implementation



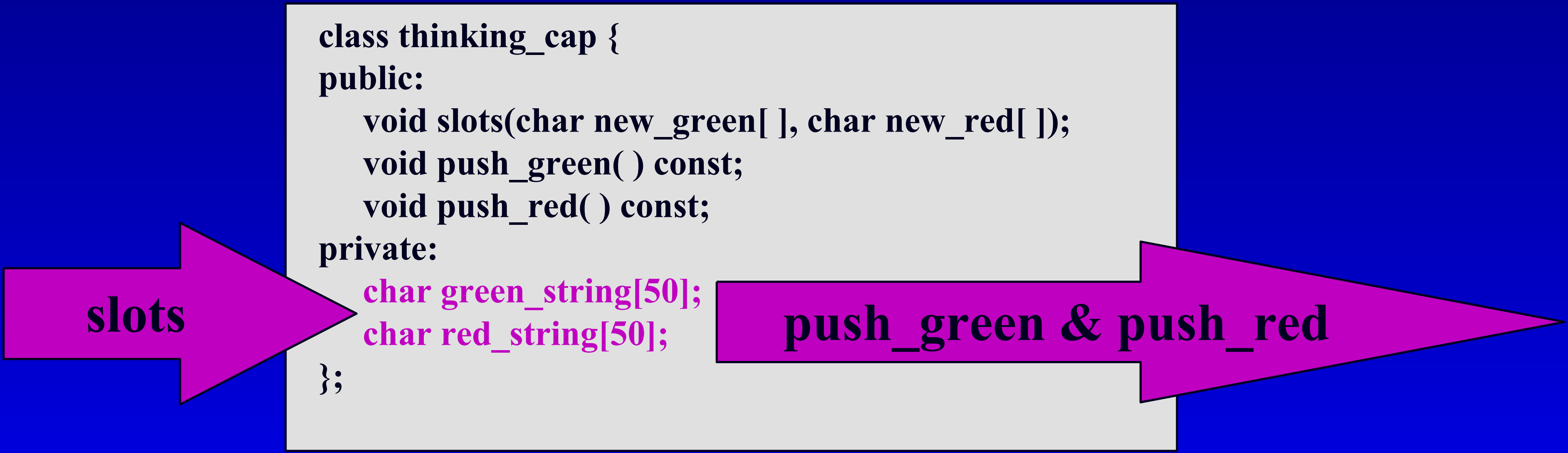
Here is the implementation of the push_green member function, which prints the green message:

```
void thinking_cap::push_green
{

    cout << green_string << endl;

}
```

Notice how this member function implementation uses the green_string member variable of the object.

A Common Pattern

- Often, one or more member functions will place data in the member variables...

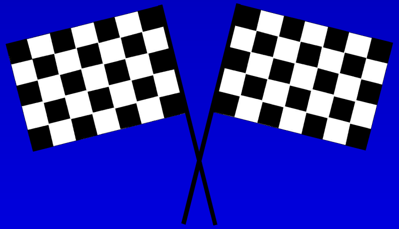


```
class thinking_cap {
public:
    void slots(char new_green[ ], char new_red[ ]);
    void push_green( ) const;
    void push_red( ) const;
private:
    char green_string[50];
    char red_string[50];
};
```

- ...so that other member functions may use that data.

Summary

- Classes have member variables and member functions. An object is a variable where the data type is a class.
- You should know how to declare a new class type, how to implement its member functions, how to use the class type.
- Frequently, the member functions of an class type place information in the member variables, or use information that's already in the member variables.
- In the future we will see more features of OOP.



THE END