

# IMPLEMENTING C++ CLASSES

---

Problem Solving with Computers-II



Read the syllabus. Know what's required. Know how to get help.

CLICKERS OUT – FREQUENCY AB

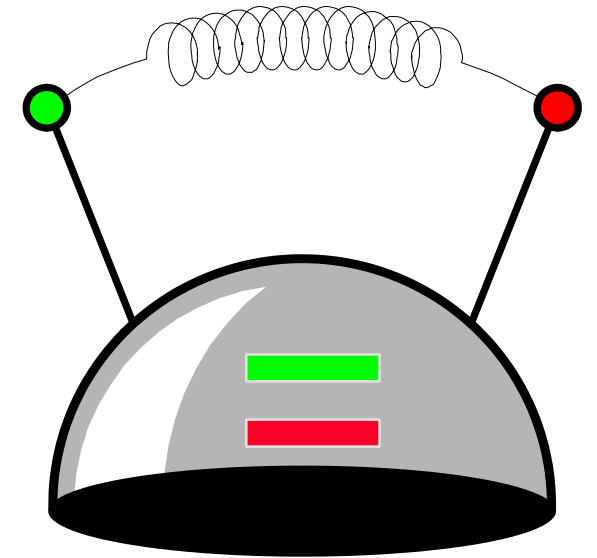
# Announcements

- Submit CS16 final exam (part of HW 1) tomorrow in sections
- If you want to pair with someone in the same section (different mentor group), let your current mentor know asap
- Mentor groups will be finalized by tomorrow.

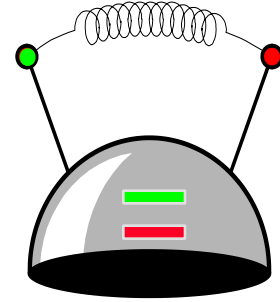
Clickers out – frequency AB

# Description of the thinking cap

- 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.



# Thinking Cap Implementation

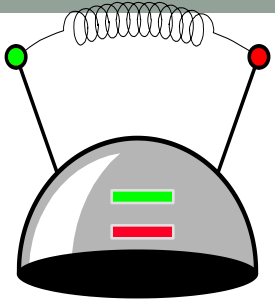


- Usually we implement the class in a separate .cpp file.

```
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];
};
```

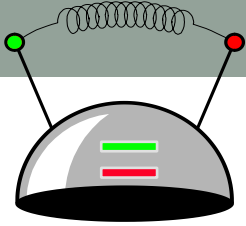
Function bodies  
will be in .cpp file.

# Thinking Cap Implementation



There are two special features about a member function's implementation . . .

```
void thinking_cap::slots(char new_green[ ], char new_red[ ])  
{  
  
  
  
  
  
  
  
  
  
}
```

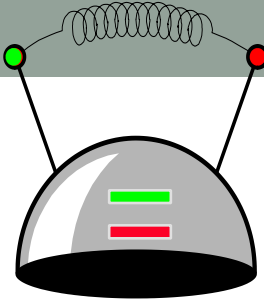


# Thinking Cap Implementation

There are two special features about a member function's implementation . . .

1. The class name is included in the function's heading using the :: operator
2. The function can refer to any of the member variables

```
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 methods 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);
}
```

*But, whose member variables are*

*these? Are they*

*c1.green\_string*

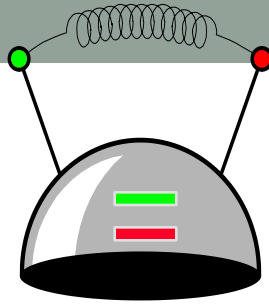
*c1.red\_string*

*c2.green\_string*

*c2.red\_string*

?

# Thinking Cap Implementation



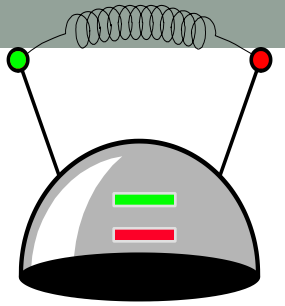
Within the body of the function, the class's member variables and other member functions may all be accessed.

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

*If we activate `c1.slots()`:*  
*`c1.green_string`*  
*`c1.red_string`*



# Thinking Cap Implementation

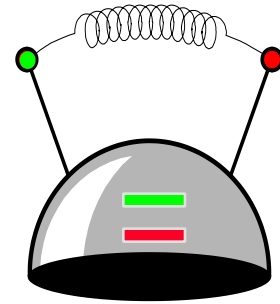


Within the body of the function, the class's member variables and other member functions may all be accessed.

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

*If we activate `c2.slots()`:*  
*`c2.green_string`*  
*`c2.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;  
  
}
```

# 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];
```

```
};
```

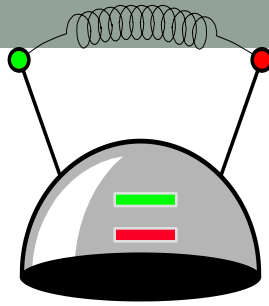


slots



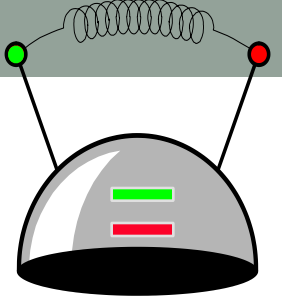
push\_green & push\_red

# Thinking Cap Definition



```
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];
};
```

- When are the data members (green\_string and red\_string) created in memory
- A. When the compiler compiles the class definition (above)
  - B. When an object of type thinking\_cap is created in the program (at run-time)
  - C. When the slots() member function is activated



# Constructor

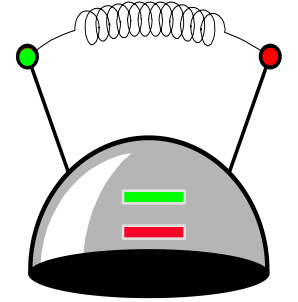
An “initialization” function that is guaranteed to be called when an object of the class is created

```
class thinking_cap
{
public:
    thinking_cap(char green[], char red[]);
    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];
};
```

*Which distinction(s) do you see between the constructor and other methods of the class?*

- A. The constructor has the same name as the class*
- B. It doesn't have a return type*
- C. It has formal parameters*
- D. A and B*
- E. None of the above*

# Implementation of the constructor

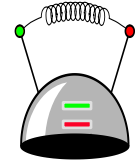


Do you expect the body of the constructor to be different from the slots() method in this example? Discuss with your group why or why not.

- A. Yes
- B. No

```
thinking_cap::thinking_cap(char green[], char red[] )  
{  
    //Code for initializing the member variables of  
  
}
```

# Using the constructor



```
class thinking_cap
{
public:
```

```
    thinking_cap(char green[], char red[]);
```

```
    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];
```

```
};
```

*What is the output of this code?*

```
int main( )
```

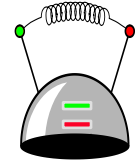
```
{
```

```
    thinking_cap c("Hi", "there");
```

```
    c.push_green( );
```

```
}
```

# Using the constructor



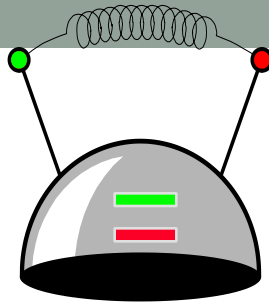
```
class thinking_cap
{
public:
    thinking_cap(char green[], char red[]);
    void slots(char new_green[ ], char new_red[ ]);
    void push_green( ) const;
    void push_red( ) const;
private:
    char green_string[50];
    char red_string[50];
};
```

*What is the output of this code?*

```
int main( )
{
    thinking_cap c;
    c.slots("Hi", "There");
    c.push_green( );
}
```



```
class thinking_cap
{
public:
    thinking_cap(); //Default constructor
    thinking_cap(char ng[], char nr[]); //Parameterized
    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];
}
```



- When are the data members (green\_string and red\_string) created in memory
- A. When the compiler compiles the class definition (above)
  - B. When an object of type thinking\_cap is created in the program (at run-time)
  - C. When the constructor explicitly creates these variables.

# Summary

- ❑ Classes have member variables and member functions (method). 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.

# Next time

- Operator overloading
- The Big four: constructor, de-structor, copy-constructor, copy-assignment