ADTs & Templates

Tiziana Ligorio
tligorio@hunter.cuny.edu

Assessment Quiz

Today's Plan

Recap

ADTs

Templates

Intro to Inheritance

Announcements and Syllabus Check

Announcements on course webpage - Questions?

Linux accounts processed this morning

Lecture slides AFTER class

Communication: csci235.help@gmail.com
(we may just skip the blackboard forum at this point)

Pointers and References (start ahead if you are not familiar with them!!!)

Syllabus: still on track, but could be running behind after today

Opportunities

Tech Talent Pipeline
https://huntercuny2x.github.io/
Application deadline soon (9/15 - 9/30):
https://cunyhunter.co1.qualtrics.com/jfe/form/SV_bNlA08EDYSsn03z

CUNY Tech Prep (for next year)
https://cunytechprep.nyc/

Recap

OPP

- Abstraction
- Encapsulation
- Information Hiding

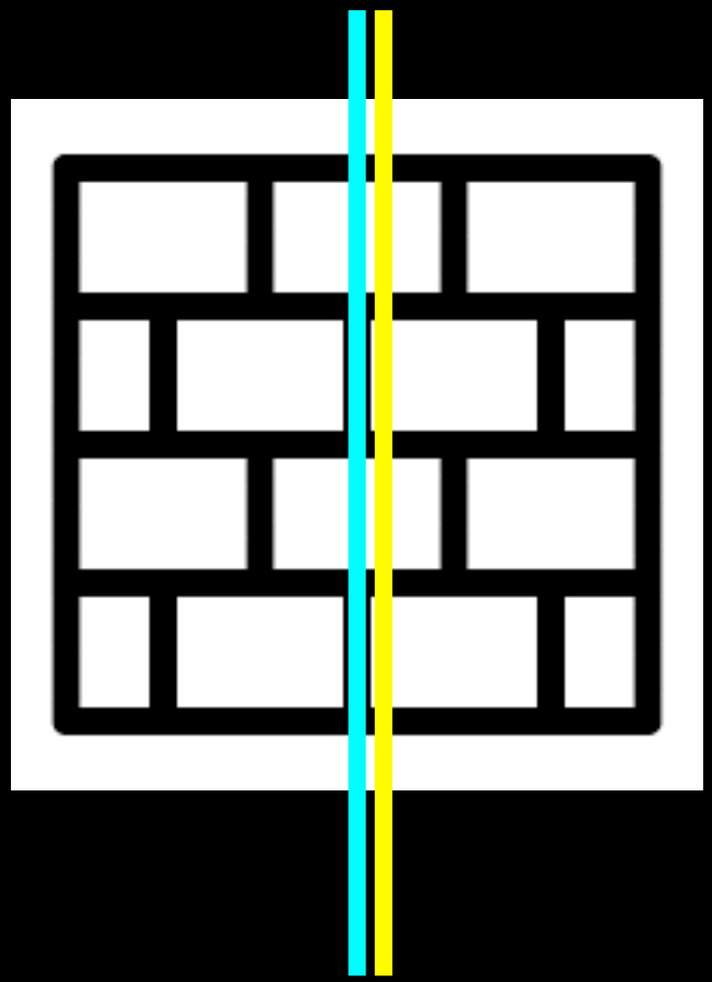
Classes

- Public Interface
- Private Implementation
- Constructors / Desctructors
- Overloading operators

Wall of Abstraction

Information barrier between device (program) use and how it works

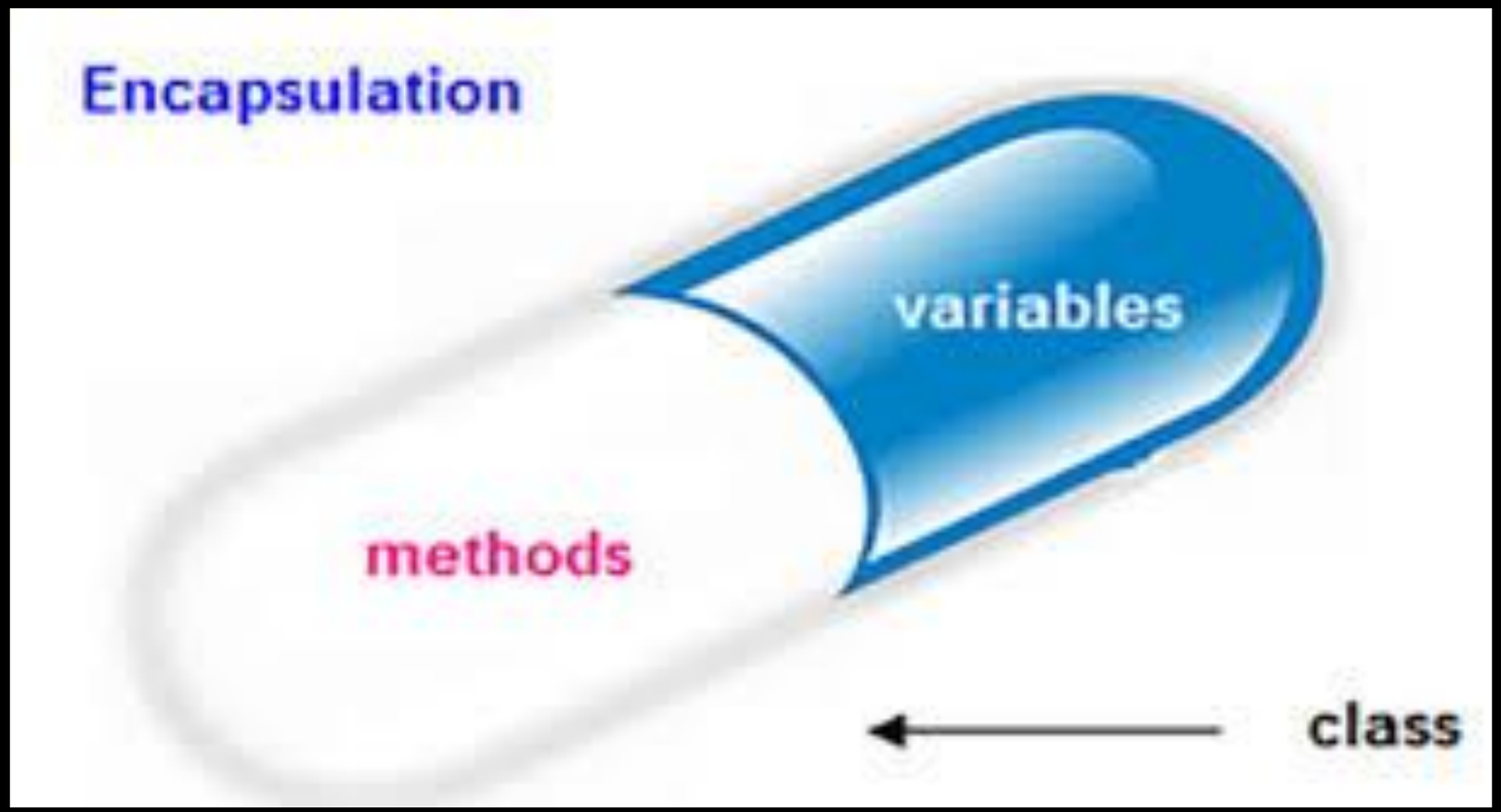
Painstaking work to design and implement stapler to work smoothly and correctly



Press handle
Or
Feed paper near sensor

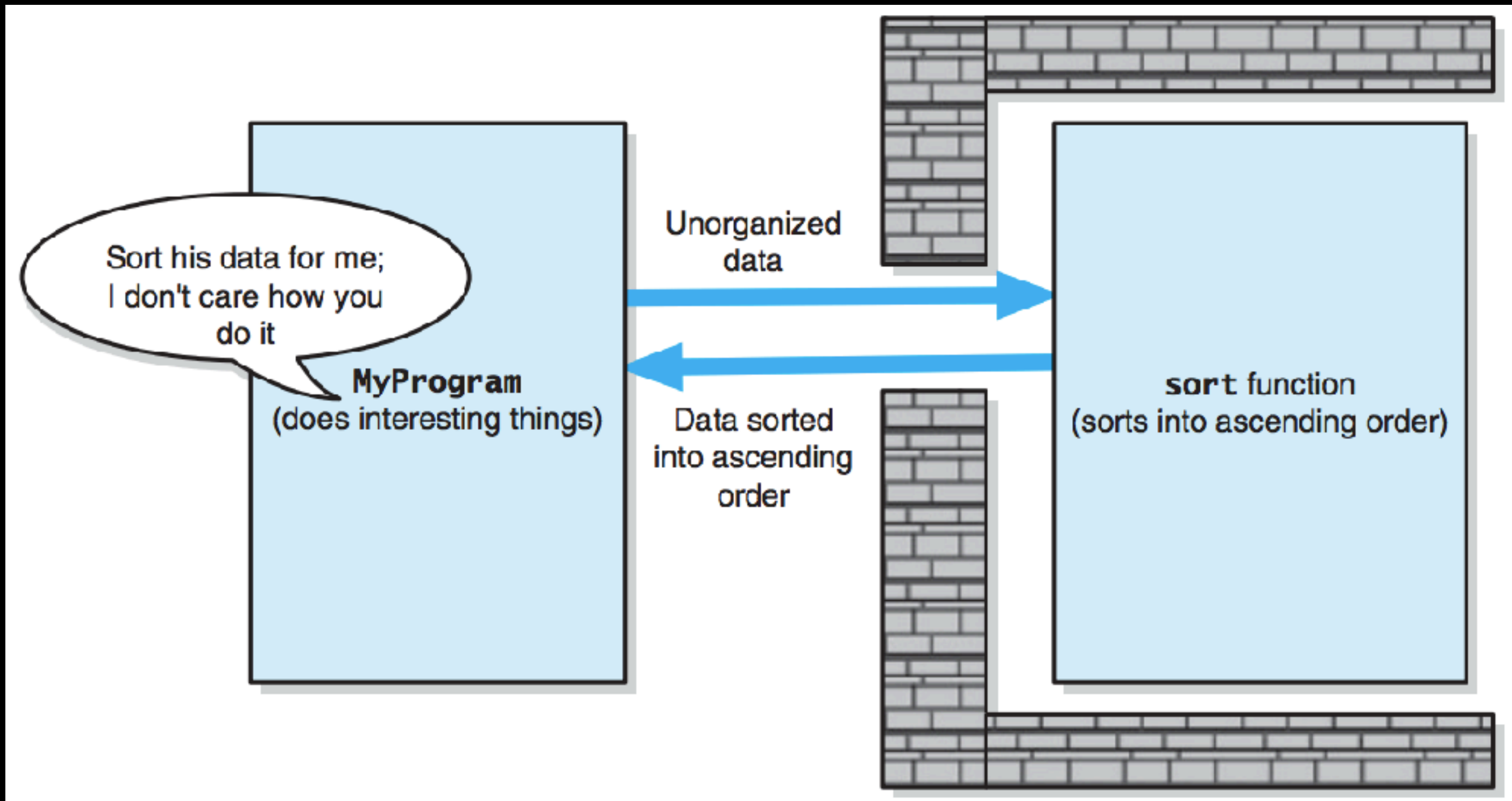
Design and implementation

Usage


Encapsulation
variables
methods
class

Class


```
class SomeClass
{
    public:
        // public data members and member functions go here

    private:
        // private data members and member functions go here

};                          // end SomeClass
```


Sort his data for me; I don't care how you do it
MyProgram
(does interesting things)
Unorganized data
Data sorted into ascending order
sort function
(sorts into ascending order)

Interface

Header file!!!!

SomeClass.h (or SomeClass.hpp)

Same as .h

Public member ***prototype*** (function declaration)

```
bool sort(int& an_array[], int number_of_elements);
```

return type + descriptive name + (parameter list)

Operation Contract

```
// these are this method's assumptions and what it does
// I will not tell you how it does it!!!
```

Operation Contract

Documents use and limitations of a method

Specifies

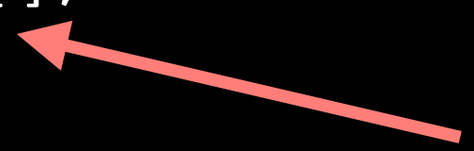
- Data flow (Input and Output)
- Pre and Post Conditions

Comments above functions in header file

Operation Contract

In Header file:

```
// sorts an array into ascending order
// pre: 1 <= number_of_elements <= MAX_ARRAY_SIZE
// post: an_array[0] <= an_array[1] <= ...
//       <= an_array[number_of_elements-1];
//       number_of_elements is unchanged
// return: true if an_array is sorted, false otherwise
bool sort(int& an_array[], int number_of_elements);
```



Function prototype

Unusual Conditions

Values out of bound, null pointer, inexistent file…

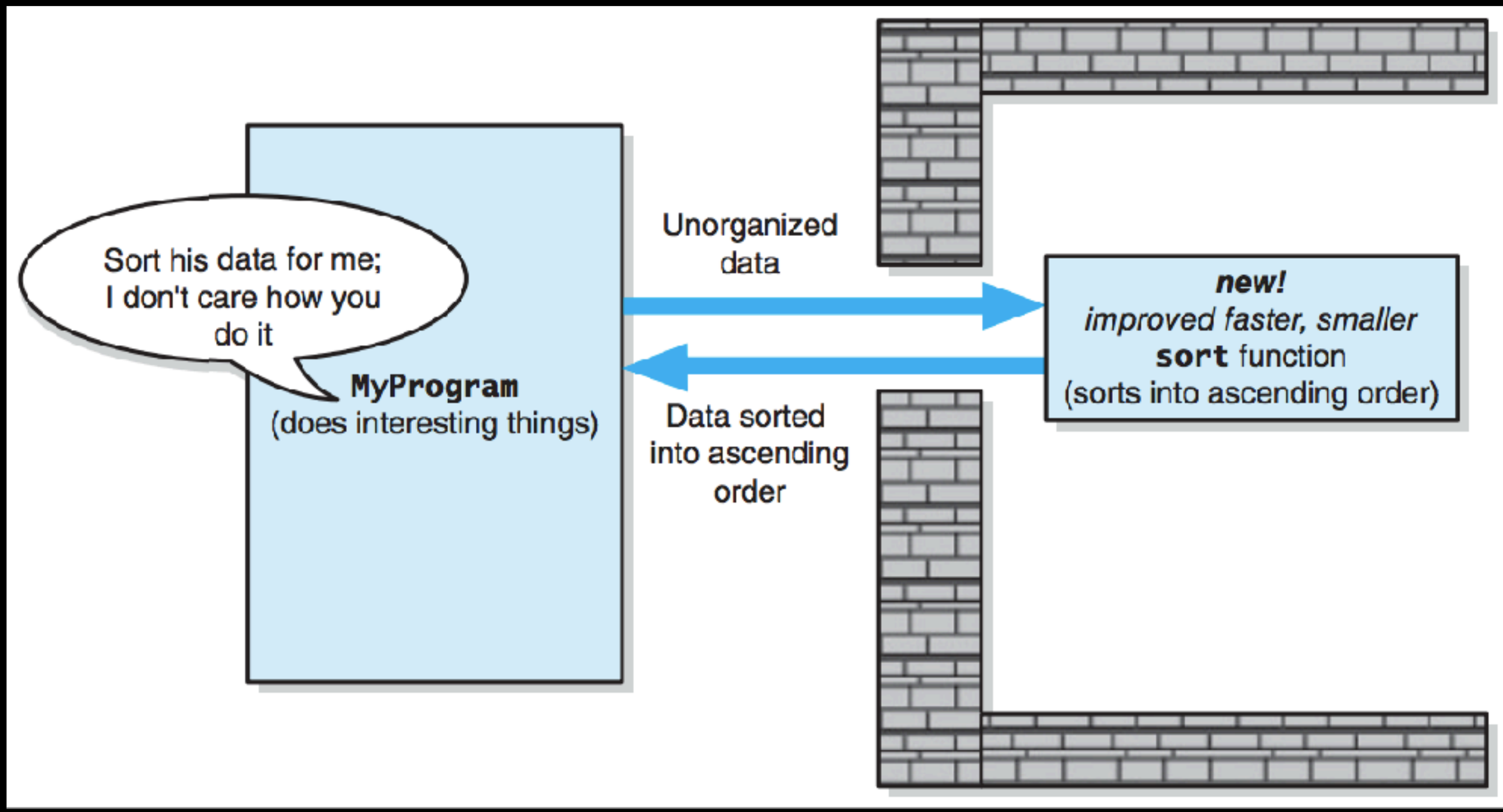
How to address them:

State it as precondition

Return value that signals a problem

Typically a boolean to indicate success or failure

Throw an exception (later in semester)


Sort his data for me;
I don't care how you
do it
MyProgram
(does interesting things)
Unorganized
data
Data sorted
into ascending
order
new!
improved faster, smaller
sort function
(sorts into ascending order)

Constructors

DECLARATION:

```
class SomeClass
{
    public:
        SomeClass();                          //default constructor
        SomeClass( parameter_list );//parameterized constructor
        // public data members and member functions go here

    private:
        // private members go here

};// end SomeClass
```

Default Constructor automatically supplied by compiler if not provided.

If only Parameterized Constructor is provided, compiler WILL NOT supply a Default Constructor and class MUST be initialized with parameters

Constructors

DECLARATION:

```
class SomeClass
{
    public:
       SomeClass();                          //default constructor
       SomeClass( parameter_list );//parameterized constructor
       // public data members and member functions go here

    private:
       // private members go here

  };// end SomeClass
```

Default Constructor automatically supplied by compiler if not provided.

If only Parameterized Constructor is provided, compiler WILL NOT supply a Default Constructor and class MUST be initialized with parameters

IMPLEMENTATION:

```
SomeClass::SomeClass()
{
}// end default constructor
```

OR:

```
SomeClass::SomeClass():
member_var1_(initial value),
member_var2_(initial value)
{
}// end default constructor
```

```
SomeClass::SomeClass(type parameter_1, type parameter_2):
member_var1(parameter_1), member_var2(parameter_2)
{
}//end parameterized constructor
```

Member Initializer List

Destructors

Default Destructors automatically supplied by compiler if not provided.

Must provide Destructor to free-up memory when SomeClass does dynamic memory allocation

```
class SomeClass
{
    public:
        SomeClass();
        SomeClass( parameter_list );//parameterized constructor
        // public data members and member functions go here
        ~SomeClass(); // destructor

    private:
        // private data members and member functions go here

};                          // end SomeClass
```

Overloading Functions

Same name, different parameter list (different function prototype)

```
int someFunction()
{

//implementation here


} // end someFunction
```

```
int someFunction(string
some_parameter )
{
    //implementation here

}   // end someFunction
```

```
int main()
{

    int x = someFunction();


    int y = someFunction(my_string);


        //more code here


} // end main
```

Friend Functions

Functions that are not members of the class but CAN access private members of the class

Violates Information Hiding!!!

Yes, so don’t do it unless appropriate and controlled

Friend Functions

DECLARATION:

```
class SomeClass
{
    public:
        // public data members and member functions go here
        friend returnType someFriendFunction( parameter list);
    private:
        // private data members and member functions go here

};                          // end SomeClass
```

IMPLEMENTATION (`SomeClass.cpp`):

Not a member function

```
returnType someFriendFunction( parameter list)
{
    // implementation here
}
```

Operator Overloading

Desirable operator (=, +, -, == …) behavior may not be well defined on objects

```
class SomeClass
{
    public:
        // public data members and member functions go here
        friend bool operator== (const SomeClass& object1,
                                const SomeClass& object2);


    private:
        // private members go here

 };    // end SomeClass
```

Operator Overloading

IMPLEMENTATION (`SomeClass.cpp`):

Not a member function

```
bool operator==(const SomeClass& object1,
                const SomeClass& object2)
{
   return ( (object1.memberA_ == object2.memberA_) &&
            (object2.memberB_ == object2.memberB_) && … );
}
```

Abstract Data Type

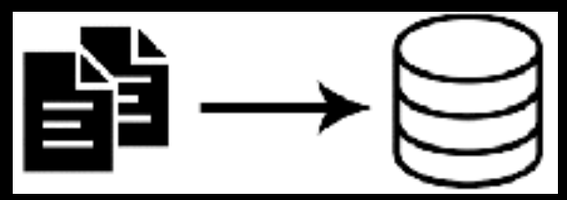
Data and Abstraction

Operations on data are central to most solutions

Think abstractly about data and its management

Typically need to

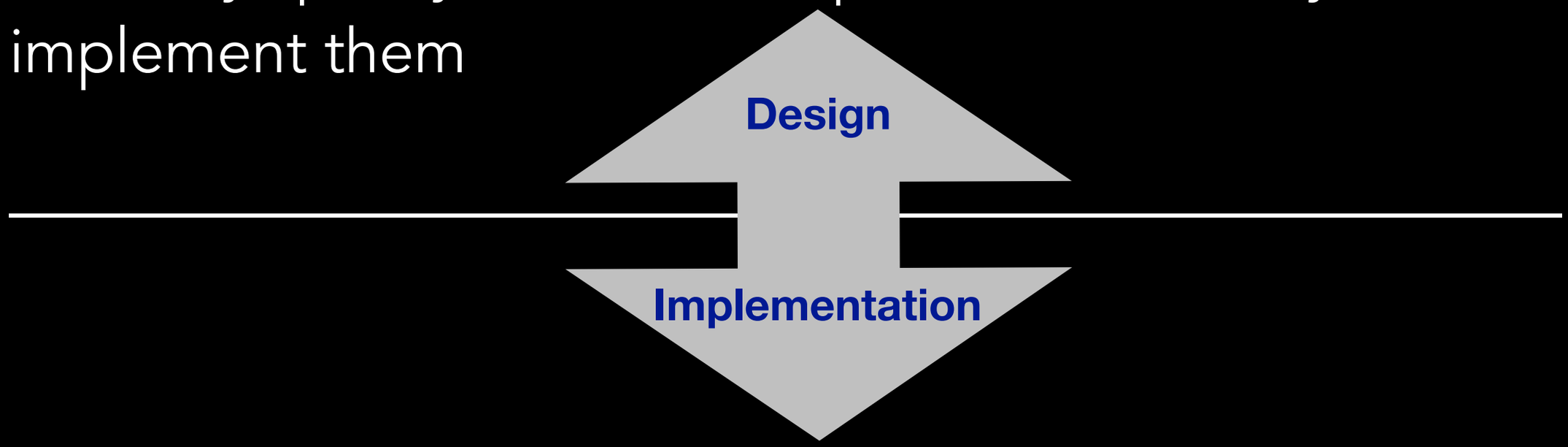
- Add data
- Remove data
- Retrieve
- Reorganize data
- Ask questions about data
- Modify data



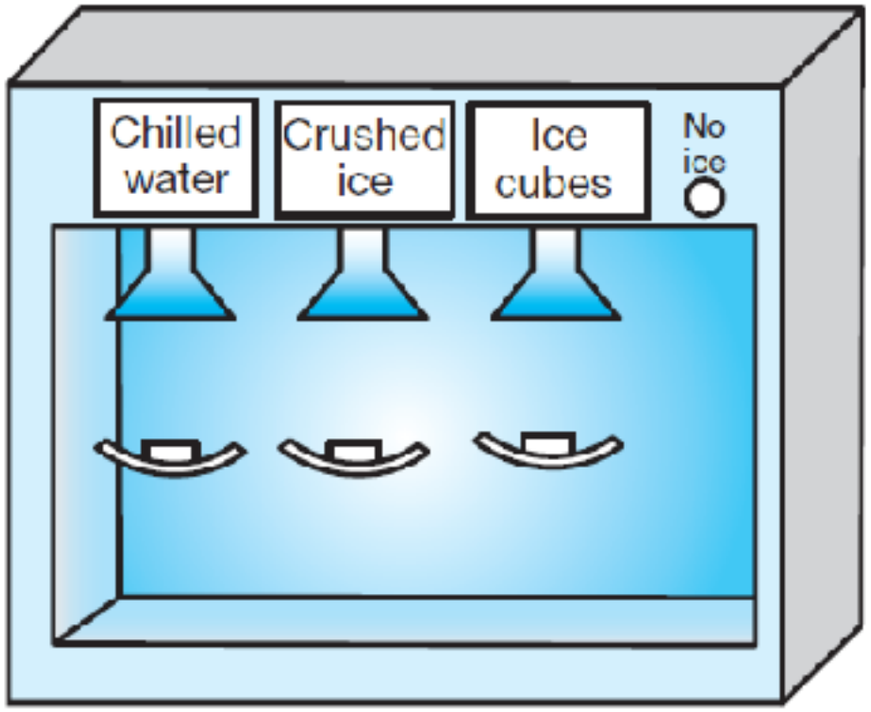
Abstract Data Type

A collection of data and a set of operations on the data

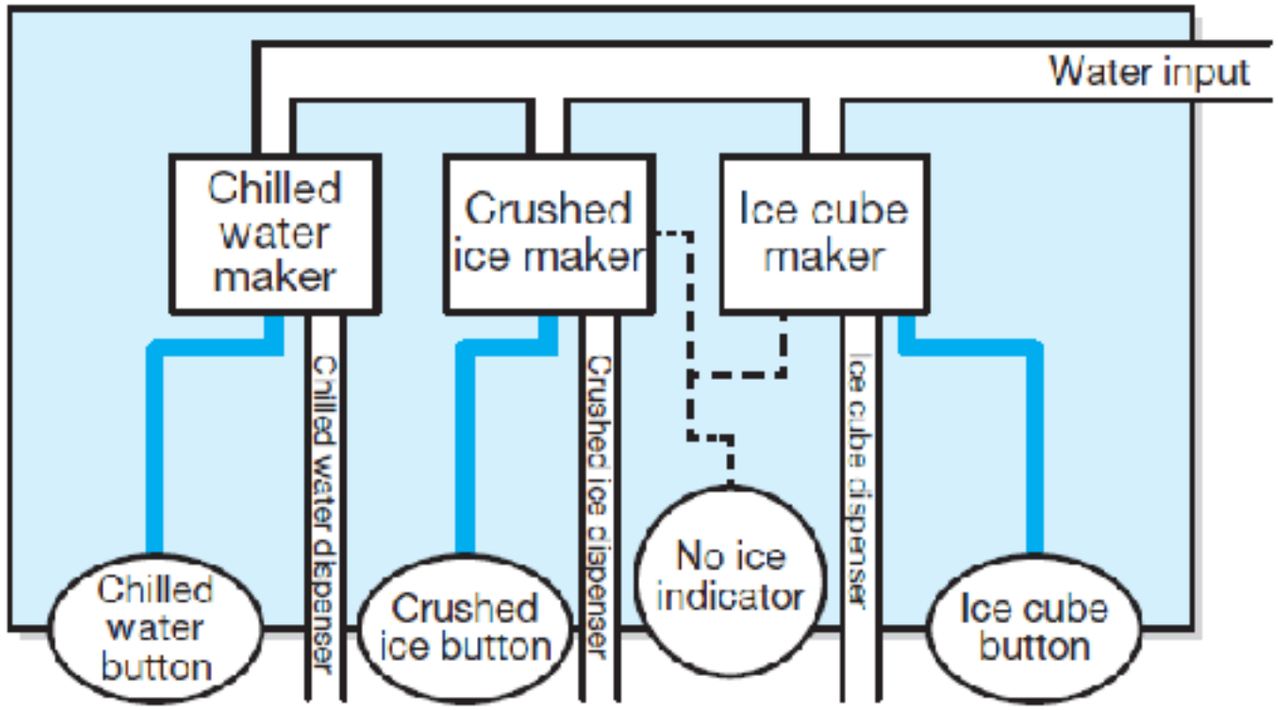
Carefully specify and ADT's operations before you implement them



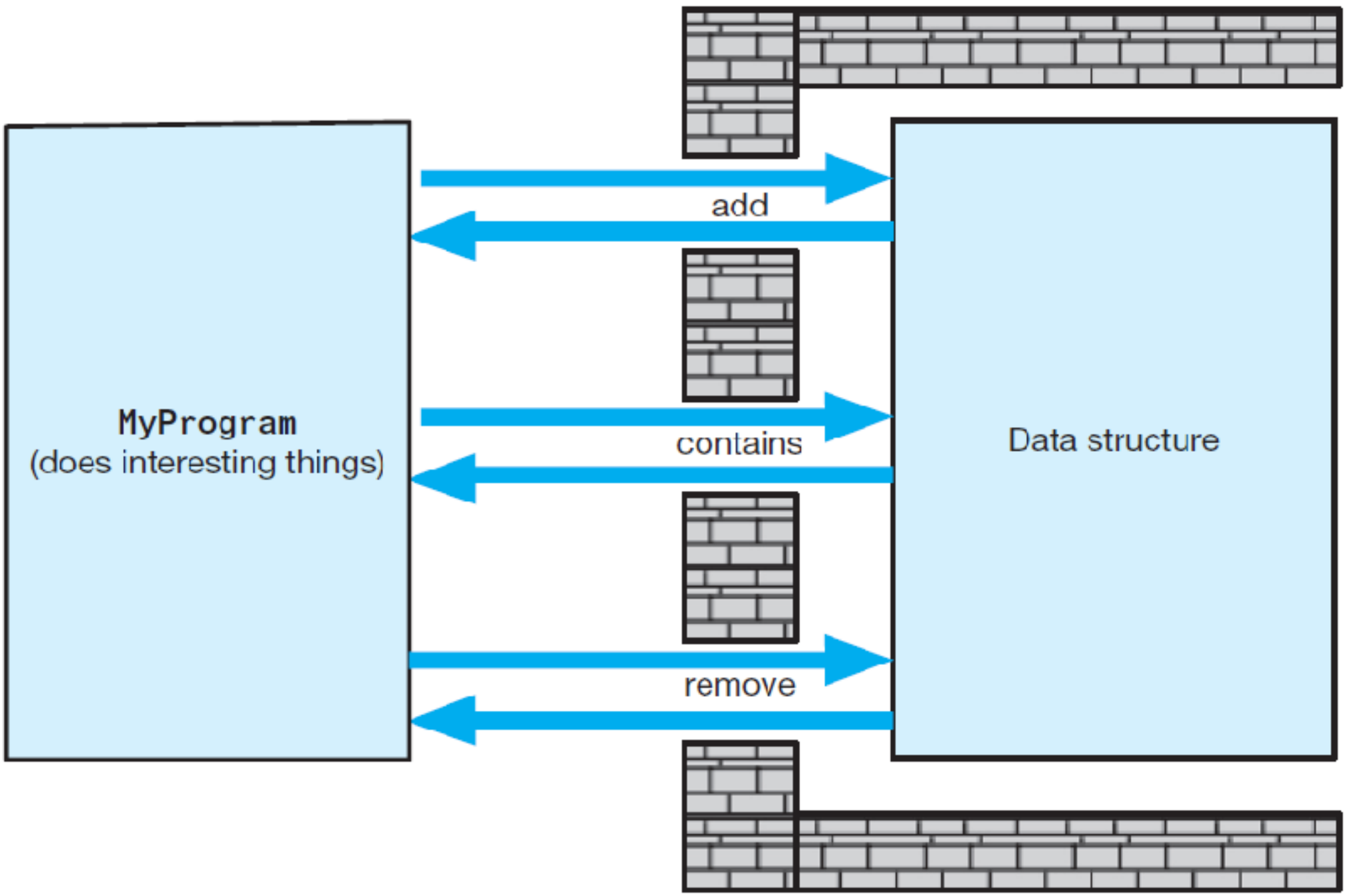
In C++ member variables and member functions implement the Data Structure



User's exterior view

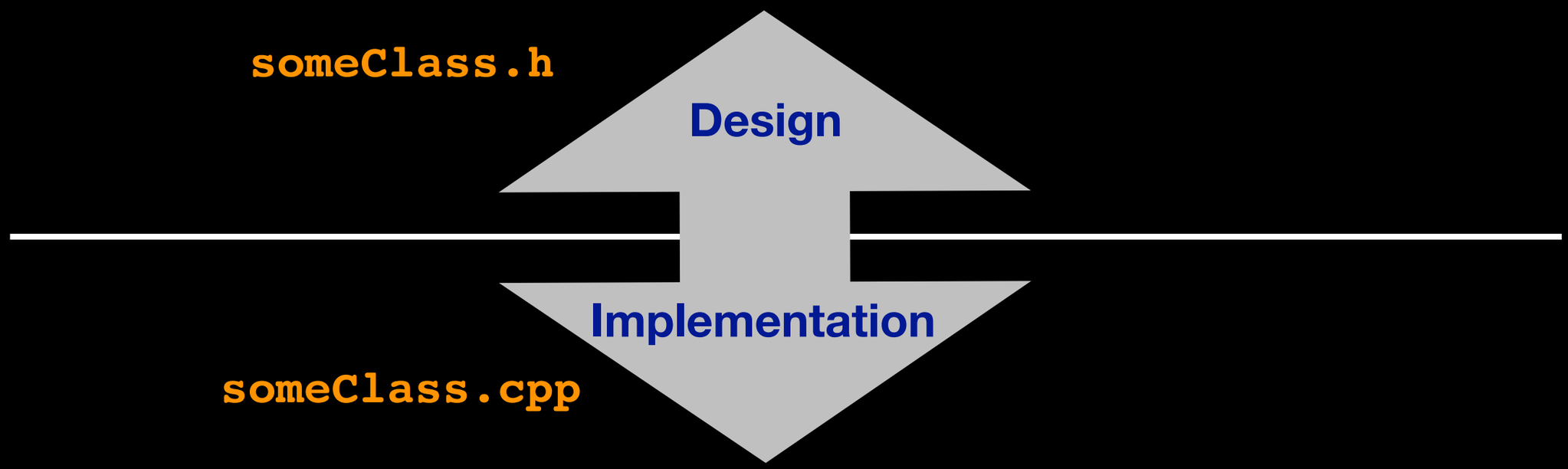


Technician's interior view


add
MyProgram
(does interesting things)
contains
Data structure
remove

Class

```
class someClass
{
    access_specifier    // can be private, public or protected
    data_members        // variables used in class
    member_functions    // methods to access data members

};                      // end someClass
```



Designing an ADT

What data does the problem require?

- Names
- IDs
- Numerical data

What operations are necessary on that data?

- Initialize
- Display
- Calculations
- Add
- Remove
- Change

Design the Bag ADT

Contains things

Container or Collection of Objects

Objects are of same type

No particular order

Can contain duplicates

In-class Task

Bag Operations:

1.
2.
3.
4.
5.
6.
…

Identify Behaviors

Bag Operations:

1. Get the number of items currently in the bag
2. See whether the bag is empty
3. Add an object to the bag
4. Remove an occurrence of a specific object form the bag, if possible
5. Remove all objects from the bag
6. Count the number of times a certain object is found in the bag
7. Test whether the bag contains a particular object
8. Look at all the objects that are in the bag

Specify Data and Operations

Pseudocode

```
//Task: reports the current number of objects in Bag
//Input: none
//Output: the number of objects currently in Bag
getCurrentSize()

//Task: checks whether Bag is empty
//Input: none
//Output: true or false according to whether Bag is empty
isEmpty()

//Task: adds a given object to the Bag
//Input: new_entry is an object
//Output: true or false according to whether addition succeeds
add(new_entry)

//Task: removes an object from the Bag
//Input: an_entry is an object
//Output: true or false according to whether removal succeeds
remove(an_entry)
```

Specify Data and Operations

```
//Task: removes all objects from the Bag
//Input: none
//Output: none
clear()

//Task: counts the number of times an object occurs in Bag
//Input: an_entry is an object
//Output: the int number of times an_entry occurs in Bag
getFrequencyOf(an_entry)

//Task: checks whether Bag contains a particular object
//Input: an_entry is an object
//Output: true of false according to whether an_entry is in Bag
contains(an_entry)

//Task: gets all objects in Bag
//Input: none
//Output: a vector containing all objects currently in Bag
toVector()
```

Vector

A container similar to a one-dimensional array

Different implementation and operations

STL (C++ standart template library)

```
#include <vector>
…
std::vector<type> vector_name;
```

e.g.

```
std::vector<string> student_names;
```

In this course cannot use STL for projects unless specified so by instructions

What's next?

Finalize the interface for your ADT => write the actual code

But we have a problem

What's next?

Finalize the interface for your ADT => write the actual code

But we have a problem

We said Bag contains objects of same type

What type?

To specify member function prototype we need to know

```
//Task: adds a given object to the Bag
//Input: new_entry is an object
//Output: true or false according to whether addition succeeds
bool add(type??? new_entry);
```

Templates

Motivation

We don’t want to write a new Bag ADT for each type of object we might want to store

Want to parameterize over some arbitrary type

Useful when implementing an ADT without locking the actual type

An example are STL containers

e.g. `vector<type>`

Declaration

```
#ifndef BAG_H_
#define BAG_H_
template<class ItemType> // this is a template definition
class Bag
{

    //class declaration here

}
#include "Bag.cpp"
#endif //BAG_H_
```

Explained next

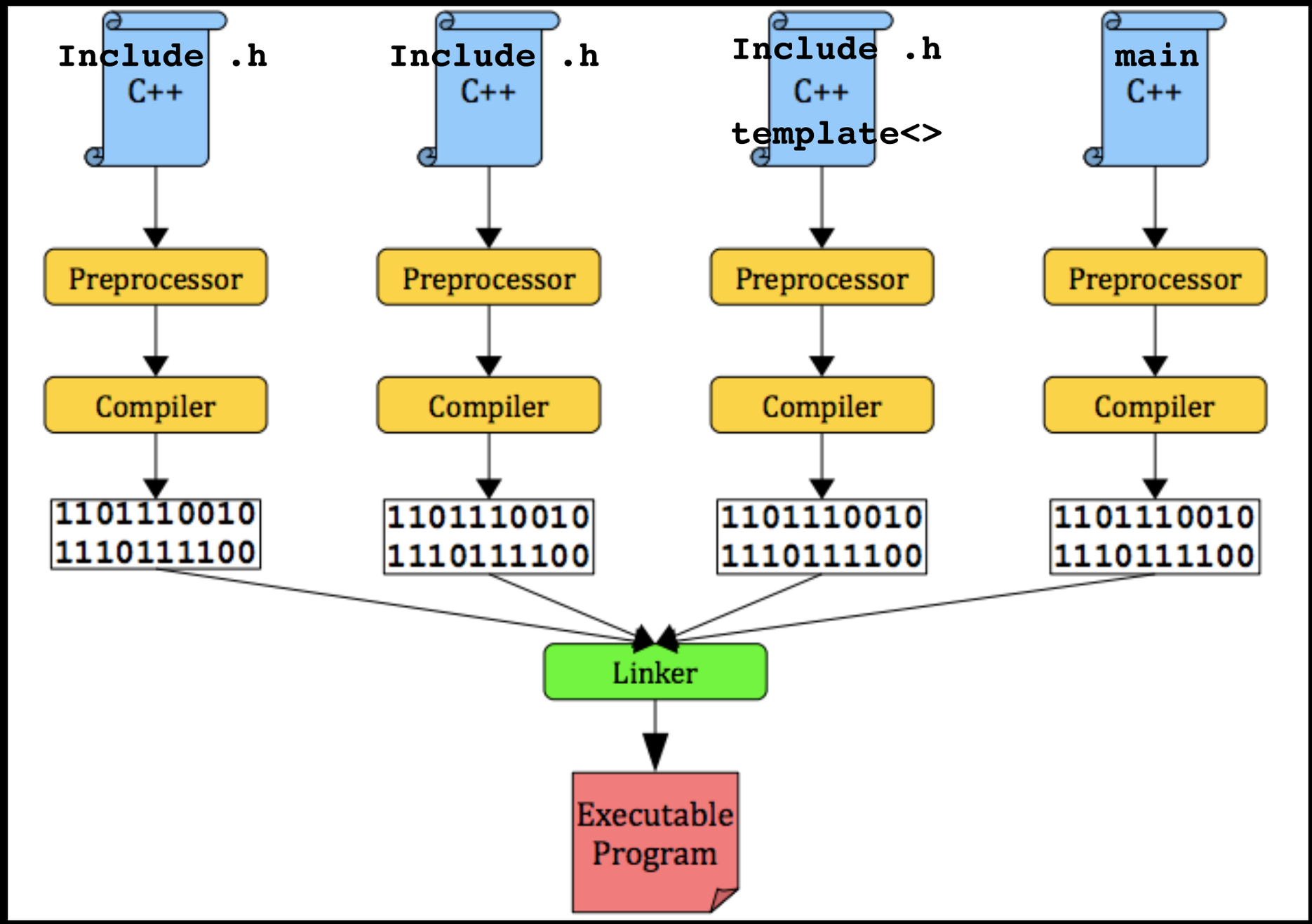
Implementation

```
#include “Bag.h”

template<class ItemType>
bool Bag<ItemType>::add(const ItemType& newEntry){
    //implementation here
}

    //more member function implementation here
```

Instantiation

```
#include “Bag.h”

int main()
{

    Bag<string> stringBag;
    Bag<int> intBag;
    Bag<someObject> someObjectBag;

    //stuff here

    return 0;

}; // end main
```

Linking with Templates



Linking with Templates

Always `#include` the `.cpp` file in the `.h` file

```
#ifndef MYCLASS_H_
#define MYCLASS_H_

//stuff here

#include “myclass.cpp”  ←
#endif //MYCLASS_H_
```

Do not add `myclass.cpp` to project and do not include it in the command to compile

```
g++ -o my_program main.cpp
```

Not

```
g++ -o my_program myclass.cpp main.cpp
```

Programming Practice

Write a simple templated dummy class `MyTemplate`

Give it some functionality, e.g:

- a parameterized constructor that initializes some private data member `my_data_` of type `ItemType`
- an accessor member function `getData()`

Write a `main()` function that initializes different `MyTemplate` objects with different types (e.g. `int`, `string`) and makes calls to their accessor member functions to observe their behavior. E.g:

```
MyTemplate<int> intObject;
cout << intObject.getData() << endl;
```

Make sure you understand and don't have problems with multi-file compilation using templates

Programming Practice

Write `BaseClass` and `DerivedClass`
(separate interface and implementation)

Play with constructors and destructors

Try different scenarios (missing `BaseClass` constructor or only parameterized `BaseClass` constructor)

Try adding `AnotherDerivedClass`

Try overloading and overriding functions

`cout<<` from every member function including constructors and destructors to trace function calls and understand behavior