Some Motivation
Some Review
Survey Responses

Big 5

Let’s look at WHY Array2D has a destructor

Pointers vs References

References: “syntactically convenient” pointers

New operator
Don’t judge yourself…

Do not judge yourself nor your ability to grasp CS concepts nor your suitability for this field based on how you’re doing in this class.

There is a LOT of material covered in this class. That’s not your imagination. This course does not define you. It’s not a predictor of your ability to learn or succeed in CS.

Keep in mind that many students have more prior knowledge and experience than you.

You have to start from where YOU are.
Grade pressures are real

I know that grade pressures are real. For some of you, they are more important than for others because of the impact they have on your ability to move forward in CS.

The LONG term is the important part for EVERYONE.

Employers don’t care as much about your grade in this class than you probably think. It also doesn’t impact your ability to get into grad school.

Knowing what you don’t know is important. Know what to do when you don’t know is MORE important.
Slow down to go faster

I understand the pressure to finish and satisfy the tests.

Trying combinations of things to solve your syntax problems is not the answer. It will usually take you MUCH longer to get your solution correct than the time you spend to do some reading and study.

I know this from firsthand experience.
Code Examples

You will find code examples online that can help you SOME.

My examples are designed along the same lines. They are not meant to be so thorough that you can copy and paste them to complete your assignment. That would defeat the purpose of the assignment.

Copy vs Move is one of those examples. The code I demonstrate only shows how copy and move are different from one another.
Example: What does “&&” mean here?

Array2DTemplate<T>::Array2DTemplate(Array2DTemplate&& arr) noexcept{}

Google: “C++ trailing &&”

https://dev.to/sandordargo/how-to-use-ampersands-in-c-3kga
How do I tell what’s useful?

Go read.

- Don’t be in a hurry.
  - The time you spend will be less than you think.
  - Time is distorted by your anxiety.
- The time you SAVE will be MORE than you think.
- The time savings keeps adding up and paying off.
- Not learning costs you time again and again.

There is still time this semester.
Don’t be in such a hurry.

Read the assignment carefully. Read it early. Think about it. Write down the scary parts.

Look at the example code and make note of anything you do not understand.

Do your reading and make notes. Include in your notes snippets that are examples of the code you need.

Try to make use of the example in a simple case in your own playground.
Variations and Permutations: Cargo Cult and other Anti-Patterns

You find yourself not understanding something in an example and just copying it to find out it doesn’t work in your context because you don’t understand a meaningful bit of syntax.

You modify the syntax until it compiles and think you’re off to the races.


I did not write my examples to satisfy your assignments. My examples are done in a vacuum. They are meant to demonstrate some portion of your assignment.

Heap vs stack example code
Don’t be in such a hurry. Read the assignment carefully.

Don’t jump right in to coding. That’s more effective when you’re much more experienced and when you’re writing things you fully understand.

Read this: “A significant amount of programming is done by superstition”

“Almost nobody can. Some people think they can, and charge forward with it, but rarely do they produce something that actually works. Usually, they bang on it and test it and cargo-cult* it until it works, then convince themselves that it's from 'first principles'.”

*https://blog.ndepend.com/cargo-cult-programming/
Don’t be in such a hurry. Read the assignment carefully.

**DO jump right in** to coding. Just don’t jump in to directly trying to satisfy the requirements of the assignment. Instead, focus on trying out the things that confuse you.

Play on your playground.

As children, we learned EVERYTHING by trial and error. All the things that we do without thinking about we learned by trial and error but we only think of classroom education as learning.

**Go play!** (and learn by doing)
Don’t be in such a hurry. Look at the example code...

- Look at the code provided and make note of anything you do not understand.
- Be honest with yourself and don’t worry if you have already forgotten or realize you don’t REALLY understand something.
- Do your reading and make notes. Include in your notes snippets that are examples of the code you need.
- Try to make use of the example in a simple case in your own playground.
- Build a document with
  - snippets of the code you don’t understand,
  - the questions you have,
  - the answers you’ve put together and
  - the links that you used to get the answers.
Driving Analogy (even if you don’t drive)

How is driving like coding?
- Unfamiliar skill
- Unfamiliar vehicle
- Unknown route to destination

Why should I slow down to get to my destination faster?

Right? What could POSSIBLY go wrong?
Example: How is clear() different from the destructor?

Google: “c++ linked list clear()”

Notice that the examples can not be readily copied and pasted to satisfy your assignment.

Not specific enough?
Example: How is clear() different from the destructor?

Google: “c++ linked list clear()”

Notice that the examples can not easily be copied and pasted to satisfy your assignment.

Not specific enough? “C++ How is clear() different from the destructor”
Use Patrick’s materials

Patrick’s slides do an excellent job of synopsizing the material that I have presented over the last couple of weeks.

If you have not already done so, go through his slides and watch his video.
Survey Responses

Big 5

Let’s look at WHY Array2D has a destructor

Pointers vs References

References: “syntactically convenient” pointers

New operator
References and Pointers

int* int_ptr;
int my_int;
int_ptr = &my_int;
int_ref& = my_int;
References and Pointers

OK, so if a reference can’t change then what are these? What’s the point?

```cpp
const int* const_int_ptr;
const int& const_int_ref;
```

https://www.learnCPP.com/cpp-tutorial/references-and-const/
Big 5 - the motivation

Let's look at WHY Array2D has a destructor which means we should do “The Big 5”
Templates

Allow us to extend functionalities of a class to a variety of types.

This is not always something we find useful but it can be in many different applications.
Templates

Templates work by using the preprocess pass of the compile/build sequence to write new code.

This is not something we witness but it helps to understand what is happening when we use a template.
Templates: an example

template <class T>
T GetMax (T a, T b) {
    T result;
    result = (a>b)? a : b;
    return (result);
}

// usage
GetMax<int> (x,y);
The Effects of: GetMax<int> (x,y);

template <class T>
T GetMax (T a, T b) {
    T result;
    result = (a>b)? a : b;
    return (result);
}

int GetMax (int a, int b) {
    int result;
    result = (a>b)? a : b;
    return (result);
}
The Effects of: GetMax<int> (x,y);

```cpp
int GetMax (int a, int b) {
    int result;
    result = (a>b)? a : b;
    return (result);
}
```
Array2D

We have built a class that dynamically allocates space for \( m \times n \) integers and allows us to store and retrieve those integers by row and column \((m, n)\) and it prevents us from reading or writing outside of the boundaries of the array.

That’s useful but this implementation is limited to integers.

It would be much more useful to be able to use the same functionality but for any type of data including user-defined types (objects).

So let’s look at Array2DTemplate*.

*this name is absurd but is meant to distinguish from the original Array2D
Array2DTemplate.h

Note that this goes in a header file.

There are a couple of ways to organize the source here by putting implementations:

- inside the class declaration.
- outside the class declaration.

I chose the latter because:

1. It was easier to convert from Array2D.cpp to template form
2. the more-cluttered syntax forces us to detangle the syntax more. In practice, I would choose the former because it’s cleaner.
Array2DTemplate.h

I chose the latter because:

1. It was easier to convert from Array2D.cpp to template form

Just insert:

```cpp
template <class T>
T* Array2DTemplate<T>::;

- OR -

template <class T>
int Array2DTemplate<T>::;
```
I chose the latter because:

1. It was easier to convert from Array2D.cpp to template form

Just insert:

```cpp
template <class T>
T* Array2DTemplate<T>::
```

*return type in red when template’s type
Array2DTemplate.h

I chose the latter because:

1. It was easier to convert from Array2D.cpp to template form

Just insert:

```cpp
template <class T>
int Array2DTemplate<T>::
```

*return type in red when NOT the template’s type*
Array2DTemplate.h

Let’s look at the code.

- Big 5 are present because of dynamic memory allocation
- `setValue();` // overloaded for both value and reference
- `floodFill();`
- Getters/setters
- `begin();`
- `end();`

`begin();` and `end();`

Only demonstrate iterator-style BEHAVIOR and syntax. It’s not suitable for your assignment and it was not meant to be.
```cpp
#include <iostream>
#include <string>
#include <vector>

template <class T>
class Container {
  public:
    Container(const T& val) : val_(val) {}
    
    void Display() const {
      std::cout << "This container contains: " << val_ << std::endl;
    }
  
  private:
    T val_;  
};

int main() {
  return 0;
}
```
Review: reference

What’s the difference when passing parameters?

- `int  int_;
- `int&  int_ref;
- `const int& const_int_ref;

Reference:

- It’s a “compiler trick”.
- Patrick’s slide calls a reference a “syntactically convenient” pointer. That’s a good way to think about it.
What’s the difference when passing parameters?

```cpp
Obj obj_;                  // a COPY of the object is passed to the function
Obj& obj_ref;             // a reference to the object is passed
const Obj& const_obj_ref; // a reference to the object is passed, the function
                           // can not modify the object
Obj&&                      // an rvalue reference
```
Review: obj_ is passed by value

What’s the difference when passing parameters?

Obj obj_; // a COPY of the object is passed to the function

You use this approach when you only need to know the value(s) of the object
Review: a reference to Obj is passed

What’s the difference when passing parameters?

Obj& obj_ref;       // a reference to the object is passed

A function is allowed to modify the parts of the object that can be modified. The function code syntactically looks as though you have the object itself.
Review: a const reference is passed

What’s the difference when passing parameters?

const Obj& const_obj_ref; // a reference to the object is passed, the function
    // can not modify the object

Why is this useful if you can’t modify?

- Large objects are not copied for passing their values.
- When you are constructing a new object, the original can not be unintentionally modified.
“unintentionally modified”

“When you are constructing a new object, the original can not be unintentionally modified.”

“Well, I have no intention of modifying it.”

This allows “unintended consequences” or “side-effects” to creep into your code. Someone (maybe even you) doesn’t know (because you forgot) that you can not modify the object.

Single Responsibility Principle
Review: Obj&& is a reference to an r-value

Reference to an rvalue.

```cpp
int&& a
```

- "a" is an r-value reference.
- && is normally only used to declare a parameter of a function.
- An r-value is a value that doesn't have a memory address.
- An r-value can not accept an assignment

```cpp
i = (y+5);     // (y+5) is an r-value
(y+5) = i;     // can’t be done, doesn’t make sense
```
Review: Obj&& is a reference to an r-value

(y+5) = i; // can’t be done, doesn’t make sense

OK, so why does THIS work?

*(y+5) = i;
(y+5) is an r-value

\[
i = (y+5);
\]

What is the lifespan of (y+5)?

“Where” is (y+5) immediately after this statement executes?

// uses move assignment operator

my_array_move_assign = Array2D("MOVE ASSIGNMENT",4,4);
What happens?

// uses move assignment operator

my_array_move_assign = Array2D("MOVE ASSIGNMENT",4,4);
What happens?

// uses move assignment operator

my_array_move_assign = Array2D("MOVE ASSIGNMENT",4,4);

Output of demo:

1. constructor with name: MOVE ASSIGNMENT
2. Move Assignment Operator from MOVE ASSIGNMENT
3. Array2D destructor: MOVE ASSIGNMENT
Array2D("MOVE ASSIGNMENT",4,4) is an r-value!

Would this make sense?

    Array2D("MOVE ASSIGNMENT",4,4) = my_array2dSecond;

Does it have an address?

It ONCE had one but it passed out of scope and no longer “exists”.