Dynamic Memory

Amrita Kaur

July 24, 2023

Contributions made from previous CS106B Instructors

Announcements

- Assignment 3 due Wednesday at 11:59pm
- Midterm Grades are out!
  - Regrade requests due Wednesday at 11:59pm

Let’s talk Midterm

Stats

- Mean: 101/120 (83.93%)
- Median: 109/120 (90.83%)
- Std dev: 22

Let’s talk Midterm

Stats

- Mean: 101/120 (83.93%)
- Median: 109/120 (90.83%)
- Std dev: 22

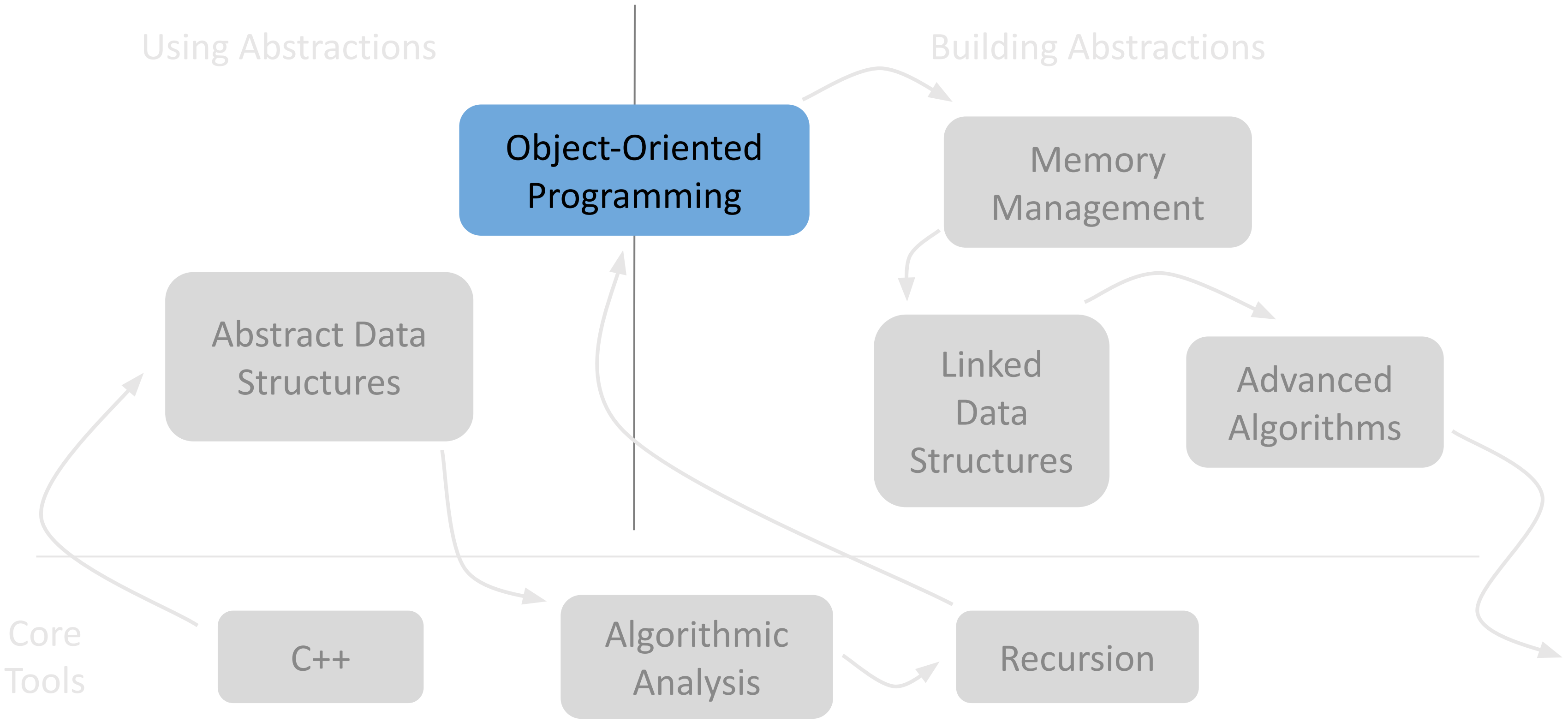
How to interpret your score:

- Between 100-120 points: Rock on! There’s always a little more to learn :)
- Between 80-99: Solid, just review those few concepts your forgot
- Below 79: Come check-in with us, now is the time to recalibrate

Regrade Requests

- Solutions are here
- If you think one of your problems was misgraded, file a regrade request on Gradescope
  - Make sure to check solutions before you submit!
  - Not for advocating for changes to the rubric itself
- If you file a request, we reserve the right to regrade the entire problem and make any necessary corrections
- Requests are due by Wednesday, July 26 at 11:59pm

Roadmap



Using Abstractions
Building Abstractions
Object-Oriented Programming
Memory Management
Abstract Data Structures
Linked Data Structures
Advanced Algorithms
Core Tools
C++
Algorithmic Analysis
Recursion

abstraction

Design that hides the details of how something works while still allowing the user to access complex functionality

Struct

- Way to bundle different types of information
  - Package data into one place
- Like **creating a custom data structure** or variable

Examples of structs

```
struct GridLocation {
   int row;
   int col;
};
```

Examples of structs

```
struct Date {
   int year;
   int month;
   int day;
};
```

Examples of structs

```
struct Lunchable {
   string dessert;
   int numCrackers;
   bool hasCheese;
};
```

Examples of structs

```
struct Album {
    string title;
    int year;

    string artist_name;
    int artist_age;
    string artist_favorite_food;
    int artist_height;
};
```

Examples of structs

```
struct Album {
    string title;
    int year;

    string artist_name;
    int artist_age;
    string artist_favorite_food;
    int artist_height;
};
```

Structs in structs

```
struct Album {
    string title;
    int year;
    Artist artist;
};
```

```
struct Artist {
    string name;
    int age;
    string favorite_food;
    int height;
};
```

Structs in structs

```
struct Album {
    string title;
    int year;
    Artist artist; ✓
};
```

```
struct Artist {
    string name;
    int age;
    string favorite_food;
    int height;
};
```

Structs in structs

```
struct Album {
    string title;
    int year;
    Album album;
};
```

```
struct Artist {
    string name;
    int age;
    string favorite_food;
    int height;
};
```

Structs in structs

```
struct Album {
    string title;
    int year;
    Album album;
};
```

✗

```
struct Artist {
    string name;
    int age;
    string favorite_food;
    int height;
};
```

```
error: field has incomplete type 'Album'
  note: definition of 'Album' is not
    complete until the closing '}'
```

Structs in structs

```
struct Album {
    string title;
    int year;
    Artist artist;
};
```

```
struct Artist {
    string name;
    int age;
    string favorite_food;
    int height;
    Album album;
};
```

Structs in structs

```
struct Album {
    string title;
    int year;
    Artist artist;
};
```

✗

```
struct Artist {
    string name;
    int age;
    string favorite_food;
    int height;
    Album album;
};
```

Class

- Defines a new data type for our program to use
- Help us create types of objects
  - Which is why we call this object-oriented programming!

Class

- Defines a **new data type** for our program to use
- Help us create types of objects
    - Which is why we call this object-oriented programming!

Struct

- Way to bundle different types of information
    - Package data into one place
- Like **creating a custom data structure** or variable

What is a class?

- The main difference between structs and classes are the encapsulation defaults
  - Struct defaults to **public** members (accessible outside the struct itself).
  - Class defaults to **private** members (accessible only inside the class implementation).

What is a class?

- The main difference between structs and classes are the encapsulation defaults
  - Struct defaults to **public** members (accessible outside the struct itself).
  - Class defaults to **private** members (accessible only inside the class implementation).
- Every class has two parts:
  - an **interface** specifying what operations can be performed on instances of the class
  - an **implementation** specifying how those operations are to be performed

Another way to think about classes…

- A blueprint for a new type of C++ **object**!
- The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

Three Main Parts

- Member variables *(What subvariables make up this new variable type?)*
  - These are the variables stored within the class
  - Usually not accessible outside the class implementation
- Member functions *(What functions can you call on a variable of this type?)*
  - Functions you can call on the object
  - Known as methods
- Constructor *(What happens when you make a new instance of this type?)*
  - Gets called when you create the object
  - Sets the initial state of each new object

Random Bags

Let’s write our first class!

Random Bag

- A random bag is a data structure similar to a stack or queue
- It supports two operations:
    - add, which puts an element into the random bag, and
    - remove random, which returns and removes a random element from the bag

Creating C++ Class

- Defining a class in C++ (typically) requires two steps:
  - Create a **header file** (typically suffixed with `.h`) describing what operations the class can perform and what internal state it needs.
  - Create an **implementation file** (typically suffixed with `.cpp`) that contains the implementation of the class.

- Clients of the class can then include (using the `#include` directive) the header file to use the class.

Header Files

RandomBag.h

What is in a header file?

What is in a header file?

```
#pragma once
```

This code is called a **preprocessor directive**. It’s used to make sure weird things don’t happen if you include the same header twice.

Preprocessor directives

- Include guards

```
#ifndef FILENAME_H
#define FILENAME_H
...
#endif /* FILENAME_H
```

Preprocessor directives

- Include guards

```
#ifndef FILENAME_H
#define FILENAME_H
...
#endif /* FILENAME_H
```

- `#pragma once`
  - Non-standard, widely supported
  - Advantages: less code, avoidance of name clashes, sometimes improvement in compilation speed
  - Disadvantages: not necessarily available in all compilers
- Read more here

What is in a header file?

```
#pragma once
```

This code is called a **preprocessor directive**. It’s used to make sure weird things don’t happen if you include the same header twice.

What is in a header file?

```
#pragma once

class RandomBag {



};
```

This is a **class definition**. We’re creating a new class called `RandomBag`. Like a `struct`, this defines the name of a new type that we can use in our programs.

When naming classes, use UpperCamelCase.

What is in a header file?

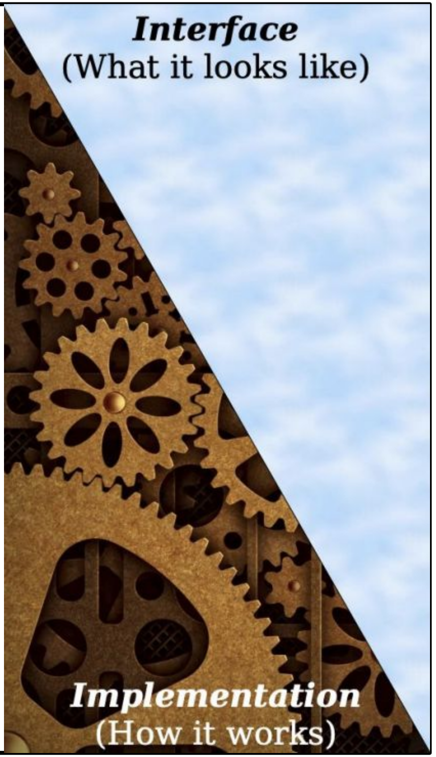
```
#pragma once

class RandomBag {




};
```

Don’t forget to add the semicolon!

You'll run into some scary compiler errors if you leave it out!

What is in a header file?

```
#pragma once

class RandomBag {
public:




private:

};
```



What is in a header file?

```
#pragma once

class RandomBag {
public:




private:

};
```

The **public interface** specifies what functions you can call on objects of this type. (i.e. its methods)

Think things like the `Vector` `.add()` function or the `string`'s `.find()`.

What is in a header file?

```
#pragma once

class RandomBag {
public:




private:

};
```

The **public interface** specifies what functions you can call on objects of this type. (i.e. its methods)

Think things like the `Vector` `.add()` function or the `string`'s `.find()`.

The **private implementation** contains information that objects of this class type will need in order to do their job properly. This is invisible to people using the class.

What is in a header file?

```
#pragma once

class RandomBag {
public:
    void add(int value);
    int removeRandom();


private:


};
```

These are **member functions** of the RandomBag class. They're functions you can call on objects of type RandomBag.

All member functions must be defined in the class definition. We'll implement these functions in the C++ file.

What is in a header file?

```
#pragma once
#include "vector.h"
class RandomBag {
public:
    void add(int value);
    int removeRandom();


private:
    Vector<int> elems;
};
```

This is a **member variable** of the class. This tells us how the class is implemented. Internally, we're going to store a `Vector<int>` holding all the elements. The only code that can access or touch this `Vector` is the `RandomBag` implementation

What is in a header file?

```
#pragma once
#include "vector.h"
class RandomBag {
public:
    void add(int value);
    int removeRandom();


private:
    Vector<int> elems;
};
```

Implementation Files

RandomBag.cpp

```
#include "RandomBag.h"
```

If we're going to implement the `RandomBag` type, the `.cpp` file needs to have the class definition available. All implementation files need to include the relevant headers.

```
#include "RandomBag.h"
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}
```

The syntax **RandomBag::add** means “the add function defined inside of RandomBag." The :: operator is called the **scope resolution operator** in C++ and is used to say where to look for things.

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void ~~RandomBag::~~add(int value){
     elems.add(value);
}
```

If we had written something like this instead, then the compiler would think we were just making a free function named add that has nothing to do with RandomBag’s version of add. That’s an easy mistake to make!

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}
```

We don't need to specify where `elems` is. The compiler knows that we're inside `RandomBag`, and so it knows that this means "the current `RandomBag`'s collection of elements."

Using the scope resolution operator is like passing in an invisible parameter to the function to indicate what the current instance is.

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}

int RandomBag::removeRandom() {
     if (elems.isEmpty()) {
          error("Aaaaahhh!");
     }
     int index = randomInteger(0, elems.size() - 1);
     int result = elems[index];
     elems.remove(index);
     return result;
}
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}

int RandomBag::removeRandom() {
     if (elems.isEmpty()) {
          error("Aaaaahhh!");
     }
     int index = randomInteger(0, elems.size() - 1);
     int result = elems[index];
     elems.remove(index);
     return result;
}
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size();
  bool isEmpty();
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}

int RandomBag::removeRandom() {
     if (elems.isEmpty()) {
          error("Aaaaahhh!");
     }
     int index = randomInteger(0, elems.size() - 1);
     int result = elems[index];
     elems.remove(index);
     return result;
}

int RandomBag::size() {
     return elems.size();
}
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size();
  bool isEmpty();
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}

int RandomBag::removeRandom() {
     if (elems.isEmpty()) {
          error("Aaaaahhh!");
     }
     int index = randomInteger(0, elems.size() - 1);
     int result = elems[index];
     elems.remove(index);
     return result;
}

int RandomBag::size() {
     return elems.size();
}

bool RandomBag::isEmpty() {
     return size() == 0;
}
```

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size();
  bool isEmpty();
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
      elems.add(value);
}

int RandomBag::removeRandom() {
      if (elems.isEmpty()) {
            error("Aaaaahhh!");
      }
      int index = randomInteger(0, elems.size() - 1);
      int result = elems[index];
      elems.remove(index);
      return result;
}

int RandomBag::size() {
      return elems.size();
}

bool RandomBag::isEmpty() {
      return size() == 0;
}
```

This code calls our own `size()` function. The class implementation can use the public interface.

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size();
  bool isEmpty();
private:
  Vector<int> elems;
};
```

```cpp
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}

int RandomBag::removeRandom() {
     if (elems.isEmpty()) {
          error("Aaaaahhh!");
     }
     int index = randomInteger(0, size() - 1);
     int result = elems[index];
     elems.remove(index);
     return result;
}

int RandomBag::size() {
     return elems.size();
}

bool RandomBag::isEmpty() {
     return size() == 0;
}
```

Let’s use it another place too!

```cpp
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size();
  bool isEmpty();
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
      elems.add(value);
}

int RandomBag::removeRandom() {
      if (elems.isEmpty()) {
            error("Aaaaahhh!");
      }
      int index = randomInteger(0, size() - 1);
      int result = elems[index];
      elems.remove(index);
      return result;
}

int RandomBag::size() {
      return elems.size();
}

bool RandomBag::isEmpty() {
      return size() == 0;
}
```

This use of the `const` keyword means "I promise that this function doesn't change the state of the object.”

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size() const;
  bool isEmpty() const;
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
      elems.add(value);
}

int RandomBag::removeRandom() {
      if (elems.isEmpty()) {
            error("Aaaaahhh!");
      }
      int index = randomInteger(0, size() - 1);
      int result = elems[index];
      elems.remove(index);
      return result;
}

int RandomBag::size() const {
      return elems.size();
}

bool RandomBag::isEmpty() const {
      return size() == 0;
}
```

We have to remember to add it to the implementation as well!

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size() const;
  bool isEmpty() const;
private:
  Vector<int> elems;
};
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
     elems.add(value);
}

int RandomBag::removeRandom() {
     if (elems.isEmpty()) {
          error("Aaaaahhh!");
     }
     int index = randomInteger(0, size() - 1);
     int result = elems[index];
     elems.remove(index);
     return result;
}

int RandomBag::size() const {
     return elems.size();
}

bool RandomBag::isEmpty() const {
     return size() == 0;
}
```

Note: There are some additional #includes that we'll need. (We'll see them in the actual .cpp file.)

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size() const;
  bool isEmpty() const;
private:
  Vector<int> elems;
};
```

Three Main Parts

- Member variables *(What subvariables make up this new variable type?)*
  - These are the variables stored within the class
  - Usually not accessible outside the class implementation
- Member functions *(What functions can you call on a variable of this type?)*
  - Functions you can call on the object
  - Known as methods
- **Constructor *(What happens when you make a new instance of this type?)***
  - **Gets called when you create the object**
  - **Sets the initial state of each new object**

Constructor

- Specially defined method for classes that initializes the state of new objects as they are created
  - Often accepts parameters for the initial state of the fields.
  - Special naming convention defined as `ClassName()`
  - You can never directly call a constructor, but one will always be called when declaring a new instance of an object

```
// MyClass.h
class MyClass {
public:


    returnType func1(parameters);
    returnType func2(parameters);
    returnType func3(parameters);

private:
    type var1;
    type var2;
    type func4();
};
```

```cpp
// MyClass.h
class MyClass {
public:
    MyClass();

    returnType func1(parameters);
    returnType func2(parameters);
    returnType func3(parameters);

private:
    type var1;
    type var2;
    type func4();
};
```

```
// MyClass.h
class MyClass {
public:
    MyClass();

    returnType func1(parameters);
    returnType func2(parameters);
    returnType func3(parameters);

private:
    type var1;
    type var2;
    type func4();
};
```

```
// MyClass.cpp
MyClass::MyClass() {
    var1 = 1;
    var2 = 1;
}


...
```

```cpp
// MyClass.h
class MyClass {
public:
    MyClass();

    returnType func1(parameters);
    returnType func2(parameters);
    returnType func3(parameters);

private:
    type var1;
    type var2;
    type func4();
};
```

```cpp
// MyClass.cpp
MyClass::MyClass() {
    var1 = 1;
    var2 = 1;
}



...
```

```cpp
// main.cpp
int main() {
    MyClass firstInstance;
}
```

```
// MyClass.h
class MyClass {
public:
    MyClass();
    MyClass(parameters);
    returnType func1(parameters);
    returnType func2(parameters);
    returnType func3(parameters);

private:
    type var1;
    type var2;
    type func4();
};
```

```
// MyClass.cpp
MyClass::MyClass() {
    var1 = 1;
    var2 = 1;
}



...
```

```
// main.cpp
int main() {
    MyClass firstInstance;
}
```

```cpp
// MyClass.h
class MyClass {
public:
    MyClass();
    MyClass(parameters);
    returnType func1(parameters);
    returnType func2(parameters);
    returnType func3(parameters);

private:
    type var1;
    type var2;
    type func4();
};
```

```cpp
// MyClass.cpp
MyClass::MyClass() {
    var1 = 1;
    var2 = 1;
}

MyClass::MyClass(parameters) {
    ...
}
...
```

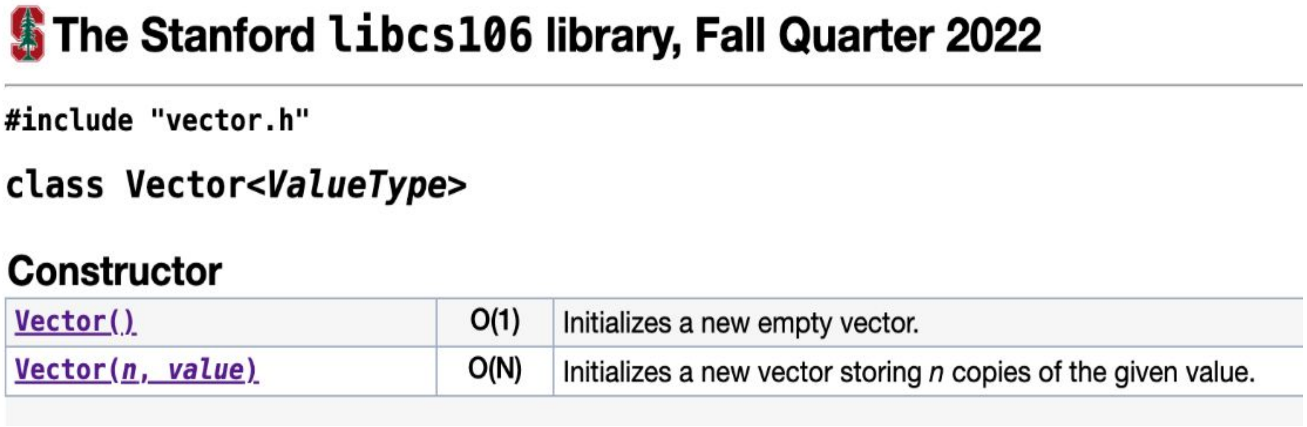
```cpp
// main.cpp
int main() {
    MyClass firstInstance;
}
```

```
// MyClass.h
class MyClass {
public:
    MyClass();
    MyClass(parameters);
    returnType func1(parameters);
    returnType func2(parameters);
    returnType func3(parameters);

private:
    type var1;
    type var2;
    type func4();
};
```

```
// MyClass.cpp
MyClass::MyClass() {
    var1 = 1;
    var2 = 1;
}

MyClass::MyClass(parameters) {
    ...
}
...
```

```
// main.cpp
int main() {
    MyClass firstInstance;
    MyClass secInstance(params);
}
```

Vector Constructors



The Stanford libcs106 library, Fall Quarter 2022

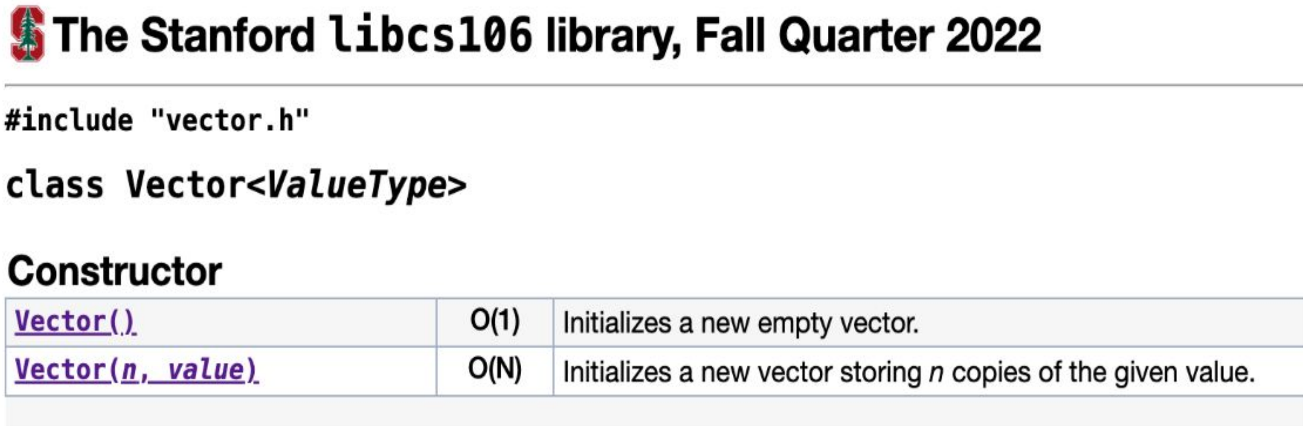
```
#include "vector.h"
```

```
class Vector<ValueType>
```

Constructor

Vector()	O(1)	Initializes a new empty vector.
Vector(*n*, *value*)	O(N)	Initializes a new vector storing *n* copies of the given value.

Vector Constructors



The Stanford libcs106 library, Fall Quarter 2022

```
#include "vector.h"
```

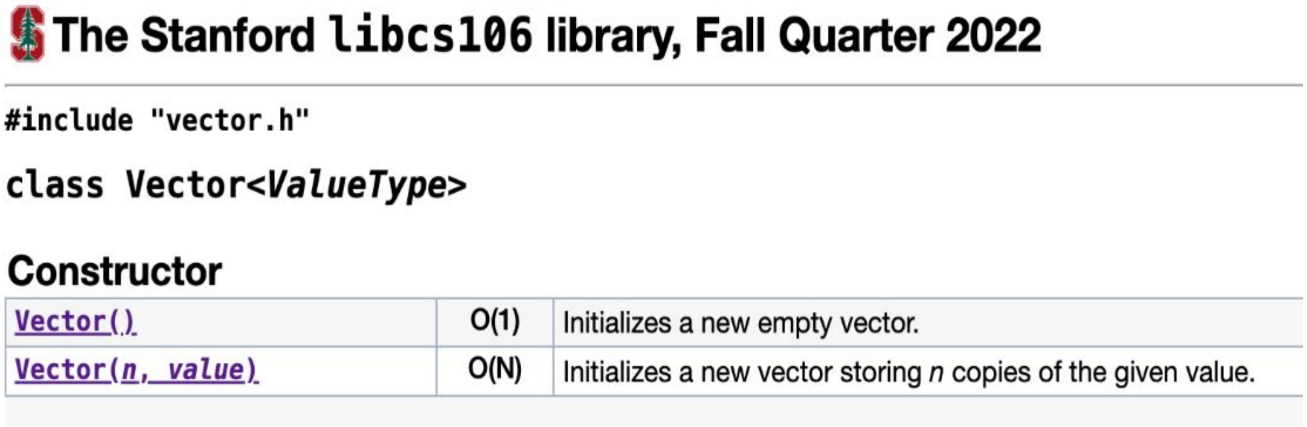
```
class Vector<ValueType>
```

Constructor

Vector()	O(1)	Initializes a new empty vector.
Vector(*n*, *value*)	O(N)	Initializes a new vector storing *n* copies of the given value.

```
Vector<string> myVec; // calls default constructor
```

Vector Constructors



The Stanford libcs106 library, Fall Quarter 2022

```
#include "vector.h"
```

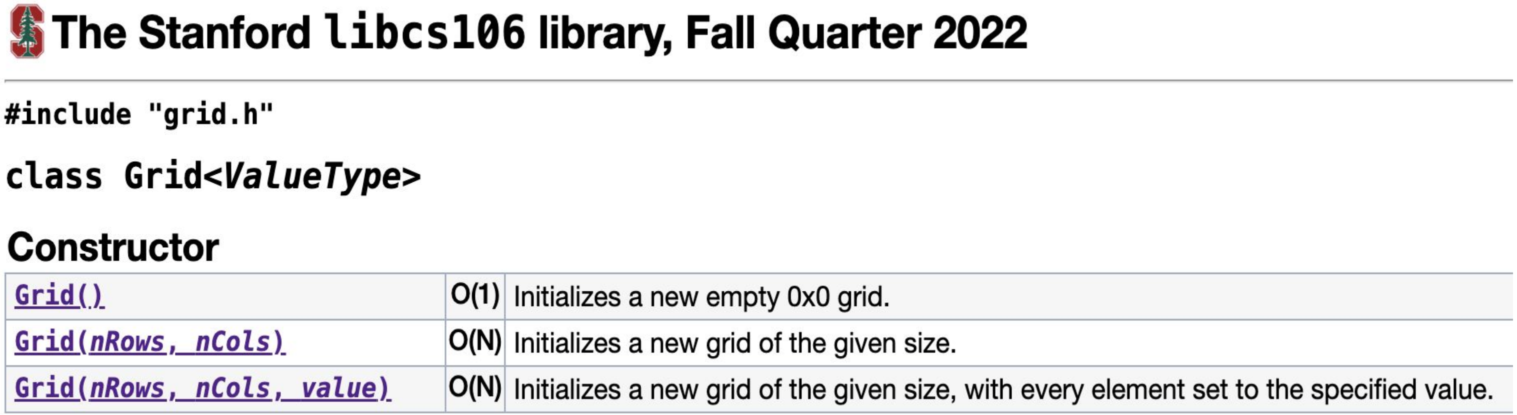
`class Vector<ValueType>`

Constructor

Vector()	O(1)	Initializes a new empty vector.
Vector(*n*, *value*)	O(N)	Initializes a new vector storing *n* copies of the given value.

```
Vector<string> myVec; // calls default constructor
Vector<string> myVec2(3, “hi”); // calls second constructor
```

Grid Constructors



The Stanford libcs106 library, Fall Quarter 2022

```
#include "grid.h"
```

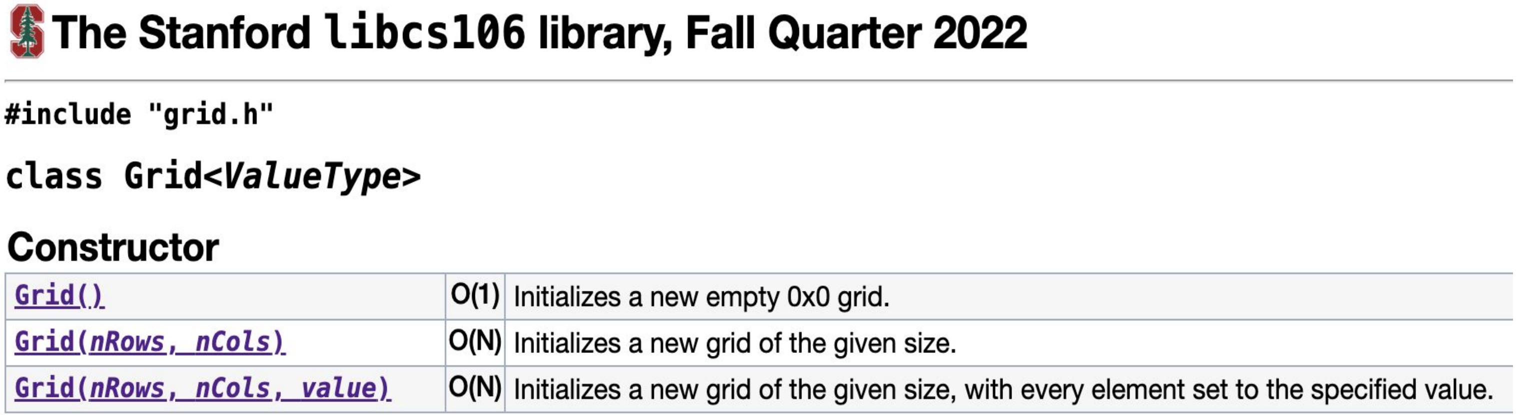
```
class Grid<ValueType>
```

Constructor

Grid()	O(1)	Initializes a new empty 0x0 grid.
Grid(nRows, nCols)	O(N)	Initializes a new grid of the given size.
Grid(nRows, nCols, value)	O(N)	Initializes a new grid of the given size, with every element set to the specified value.

```
Grid<int> myGrid; // calls default constructor
```

Grid Constructors



The Stanford libcs106 library, Fall Quarter 2022

```
#include "grid.h"
```

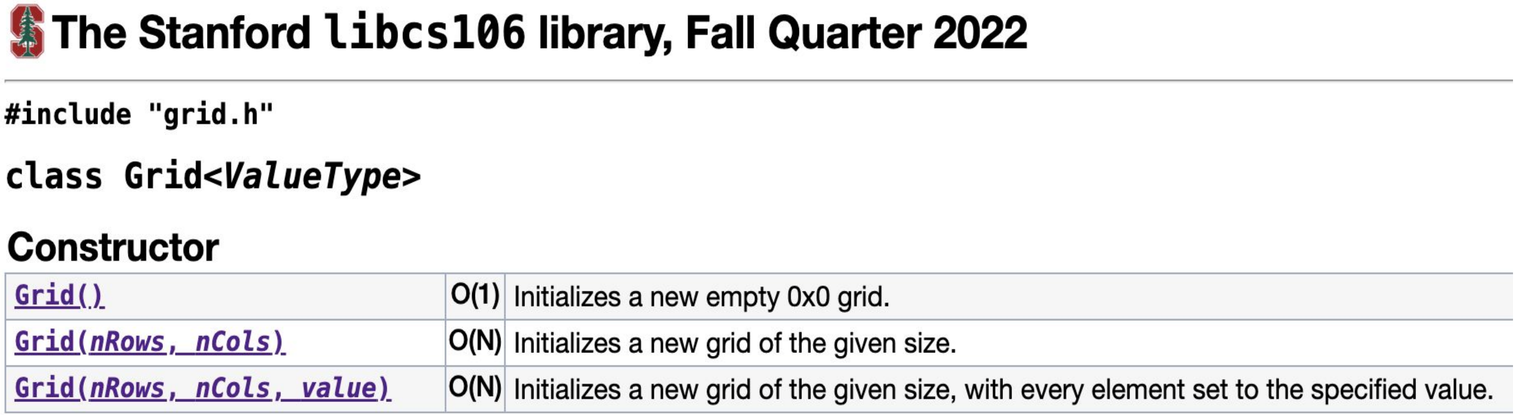
```
class Grid<ValueType>
```

Constructor

Grid()	O(1)	Initializes a new empty 0x0 grid.
Grid(nRows, nCols)	O(N)	Initializes a new grid of the given size.
Grid(nRows, nCols, value)	O(N)	Initializes a new grid of the given size, with every element set to the specified value.

```
Grid<int> myGrid; // calls default constructor
Grid<int> myGrid2(3, 4); // calls second constructor
```

Grid Constructors



The Stanford `libcs106` library, Fall Quarter 2022

```
#include "grid.h"
```

```
class Grid<ValueType>
```

Constructor

Grid()	O(1)	Initializes a new empty 0x0 grid.
Grid(*nRows*, *nCols*)	O(N)	Initializes a new grid of the given size.
Grid(*nRows*, *nCols*, *value*)	O(N)	Initializes a new grid of the given size, with every element set to the specified value.

```
Grid<int> myGrid; // calls default constructor
Grid<int> myGrid2(3, 4); // calls second constructor
Grid<int> myGrid3(3, 4, 0); // calls third constructor
```

```
#include "RandomBag.h"

void RandomBag::add(int value){
      elems.add(value);
}

int RandomBag::removeRandom() {
      if (elems.isEmpty()) {
            error("Aaaaahhh!");
      }
      int index = randomInteger(0, size() - 1);
      int result = elems[index];
      elems.remove(index);
      return result;
}

int RandomBag::size() const {
      return elems.size();
}

bool RandomBag::isEmpty() const {
      return size() == 0;
}
```

There is no explicit constructor for this class, which is okay! Instead, there's a default, zero-argument constructor that instantiates all private member variables.

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size() const;
  bool isEmpty() const;
private:
  Vector<int> elems;
};
```

```
// MyClass.h
class MyClass {
public:
    MyClass();
    MyClass(int var1, int var2);
    returnType func1(parameters);
    returnType func2(parameters);
    returnType func3(parameters);

private:
    int var1;
    int var2;
    type func4();
};
```

```
// MyClass.cpp
MyClass::MyClass() {
    var1 = 1;
    var2 = 1;
}

MyClass::MyClass(parameters) {
    ...
}
...
```

```cpp
// MyClass.h
class MyClass {
public:
    MyClass();
    MyClass(int var1, int var2);
    returnType func1(parameters);
    returnType func2(parameters);
    returnType func3(parameters);

private:
    int var1;
    int var2;
    type func4();
};
```

```cpp
// MyClass.cpp
MyClass::MyClass() {
    var1 = 1;
    var2 = 1;
}

MyClass::MyClass(int var1, int var2) {
    ...
}
...
```

```
// MyClass.h
class MyClass {
public:
    MyClass();
    MyClass(int var1, int var2);
    returnType func1(parameters);
    returnType func2(parameters);
    returnType func3(parameters);

private:
    int var1;
    int var2;
    type func4();
};
```

```
// MyClass.cpp
MyClass::MyClass() {
    var1 = 1;
    var2 = 1;
}

MyClass::MyClass(int var1, int var2) {
    this->var1 = var1;
    this->var2 = var2;
}
...
```

`this`

- Refers to the current instance of an object that a method is being called on
- Similar to the `self` keyword in Python and the `this` keyword in Java
- Syntax: `this->member`
- Common usage: In the constructor, so parameter names can match the names of the object's member variables
- `this` uses `->` not `.` because it is a *pointer*

RandomBag Code

Takeaways

- Public member variables declared in the header file are automatically accessible in the `.cpp` file.

Takeaways

- Public member variables declared in the header file are automatically accessible in the `.cpp` file.
- As a best practice, member variables should be private, and you can create public member functions to allow users to edit them

Takeaways

- Public member variables declared in the header file are automatically accessible in the `.cpp` file.
- As a best practice, member variables should be private, and you can create public member functions to allow users to edit them
- Member functions have an implicit parameter that allows them to know what instance of the class (i.e. which object) they're operating on

Takeaways

- Public member variables declared in the header file are automatically accessible in the `.cpp` file.
- As a best practice, member variables should be private, and you can create public member functions to allow users to edit them
- Member functions have an implicit parameter that allows them to know what instance of the class (i.e. which object) they're operating on
- When you don't have a constructor, there's a default, zero-argument constructor that instantiates all private member variables

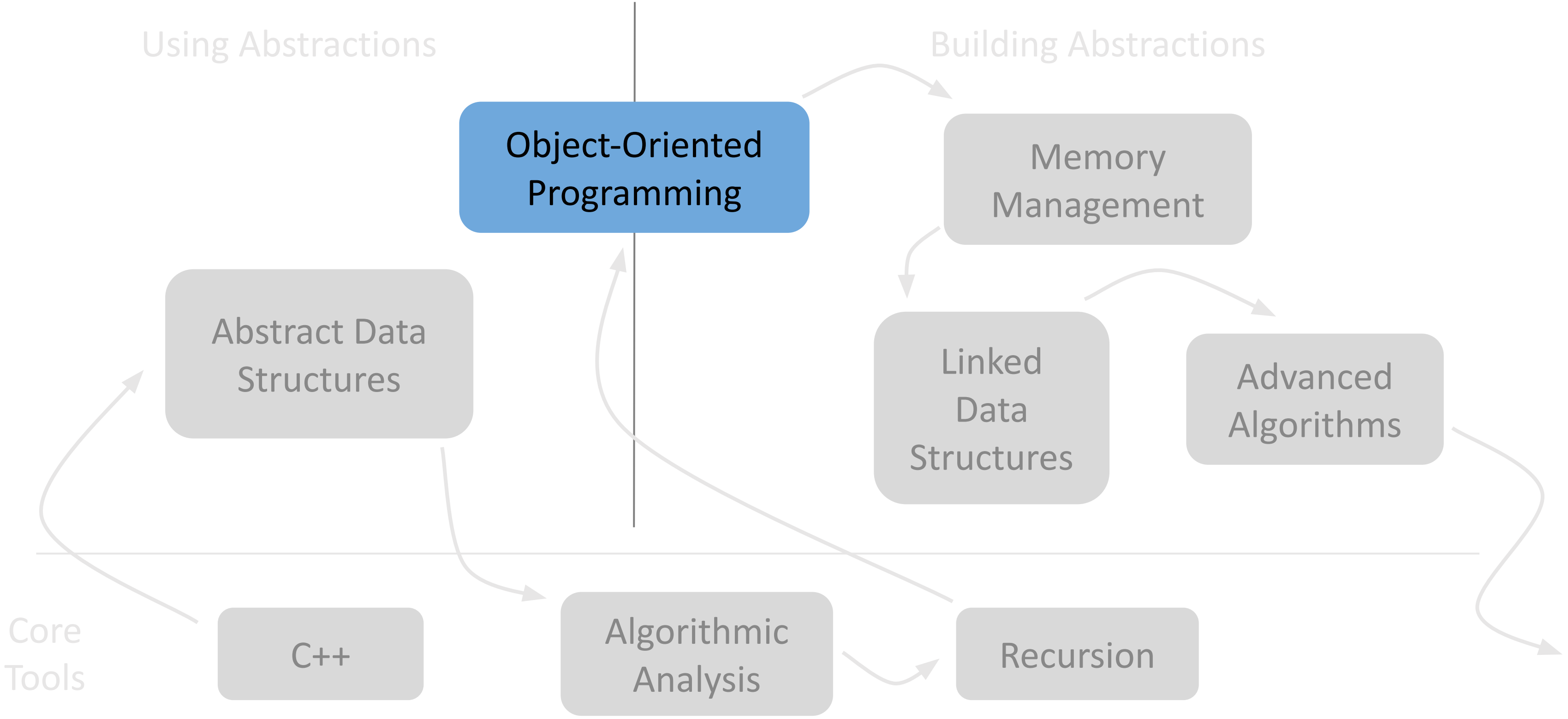
BankAccount Code

Structs vs Classes

Recap

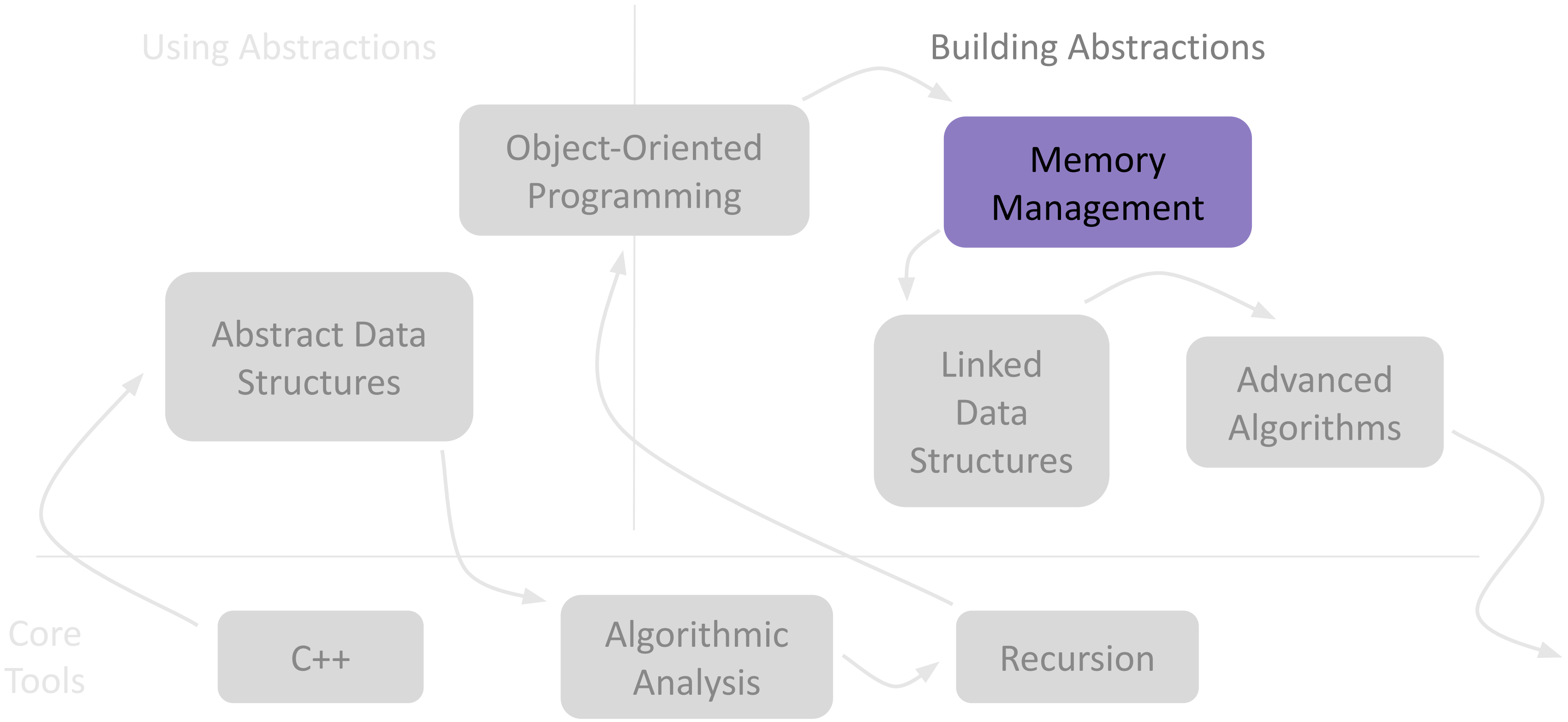
- We can create our own abstractions for defining data types using classes. Classes allow us to encapsulate information in a structured way.
- Classes have three main parts to keep in mind when designing them:
  - Member variables → these are always private
  - Member functions (methods) → these can be private or public
  - Constructor → this is created by default if you don’t define one
- Writing classes requires the creation of a header (.h) file for the interface and an implementation (.cpp) file.

Roadmap



Using Abstractions
Building Abstractions
Object-Oriented Programming
Memory Management
Abstract Data Structures
Linked Data Structures
Advanced Algorithms
Core Tools
C++
Algorithmic Analysis
Recursion

Roadmap



Using Abstractions
Building Abstractions
Object-Oriented Programming
Memory Management
Abstract Data Structures
Linked Data Structures
Advanced Algorithms
Core Tools
C++
Algorithmic Analysis
Recursion

Readymade containers are great!

- You can do so much with the ADTs that you have!
  - Write code that sorts names in the U.S. census
  - Use vectors, grids to search for optimal paths in a maze
  - Generate combinations recursively using sets
- You used their interfaces

But how are those containers implemented?

- We’ll need to learn about more basic building blocks in C++: arrays, pointers
- Tomorrow, we’re building our own vector!

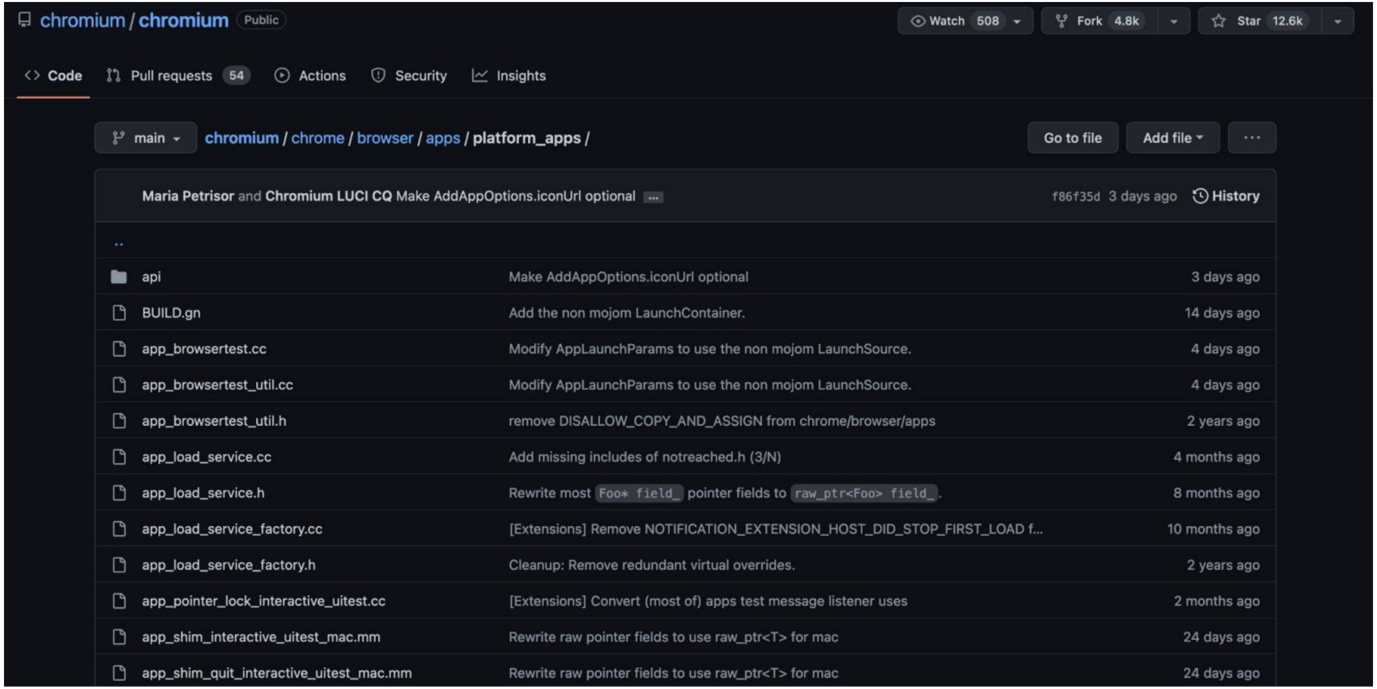
But how are those containers implemented?

- We’ll need to learn about more basic building blocks in C++: arrays, pointers
- Tomorrow, we’re building our own vector!

And what if we need custom containers / objects?

- We have to define our own classes
- In A4, you’ll be building a priority queue class!

For example, Google Chrome



Maria Petrisor and Chromium LUCI CQ Make AddAppOptions.iconUrl optional ... f86f35d 3 days ago

..		
api	Make AddAppOptions.iconUrl optional	3 days ago
BUILD.gn	Add the non mojom LaunchContainer.	14 days ago
app_browsertest.cc	Modify AppLaunchParams to use the non mojom LaunchSource.	4 days ago
app_browsertest_util.cc	Modify AppLaunchParams to use the non mojom LaunchSource.	4 days ago
app_browsertest_util.h	remove DISALLOW_COPY_AND_ASSIGN from chrome/browser/apps	2 years ago
app_load_service.cc	Add missing includes of notreached.h (3/N)	4 months ago
app_load_service.h	Rewrite most `Foo* field_` pointer fields to `raw_ptr<Foo> field_`.	8 months ago
app_load_service_factory.cc	[Extensions] Remove NOTIFICATION_EXTENSION_HOST_DID_STOP_FIRST_LOAD f...	10 months ago
app_load_service_factory.h	Cleanup: Remove redundant virtual overrides.	2 years ago
app_pointer_lock_interactive_uitest.cc	[Extensions] Convert (most of) apps test message listener uses	2 months ago
app_shim_interactive_uitest_mac.mm	Rewrite raw pointer fields to use raw_ptr<T> for mac	24 days ago
app_shim_quit_interactive_uitest_mac.mm	Rewrite raw pointer fields to use raw_ptr<T> for mac	24 days ago

Squares

- Let’s say we want to write a function `squares` that accepts an integer and creates a Vector of integers that contains all perfect squares, up to and including the square of the input
- Ex:
    - Input integer: 4
    - Output Vector: `{1, 4, 9, 16}`

Squares

```
Vector<int> squares(int numSquares) {
    Vector<int> vec;
    for (int i = 0; i < numSquares; i++) {
        vec.add(i * i);
    }
    return vec;
}
```

Squares, Take 2

```
void squares(Vector<int>& vec, int numSquares) {
    for (int i = 0; i < numSquares; i++) {
        vec.add(i * i);
    }
}
```

Squares, Take 3

```
Vector<int>& squares(int numSquares) {
    Vector<int> vec;
    for (int i = 0; i < numSquares; i++) {
        vec.add(i * i);
    }
    return vec;
}
```

What do we want?

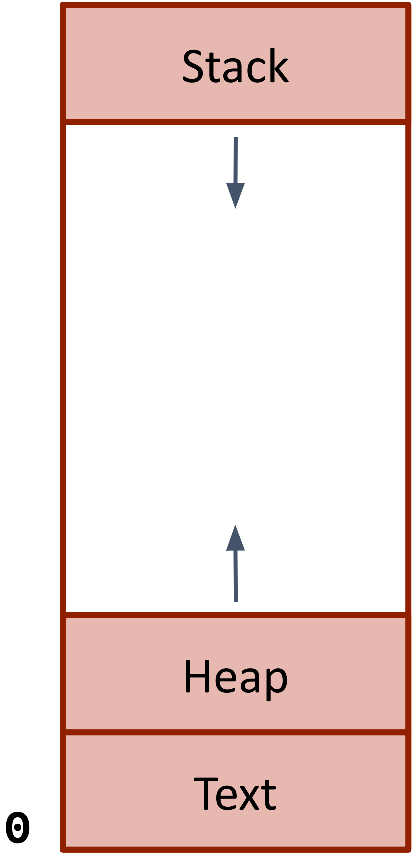
1. a way to reserve a section of memory so that it remains available to us throughout our entire program, or until we want to destroy it
2. a way to reserve any amount of memory we want at the time we need it

Global Variables

- Can be accessed by any function in our program
  - That isn't what we want
  - Want to control which function has access to the data, just like we normally would when passing data between functions
- Have a fixed size at compile time
  - That isn't what we want.

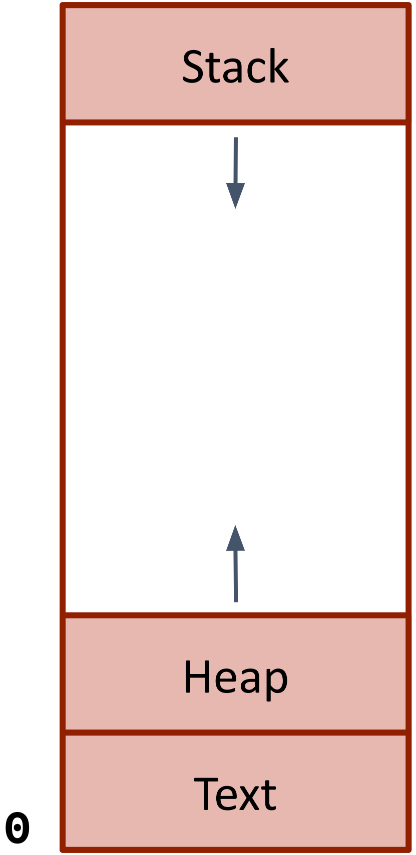
Dynamic Memory Allocation

- Use dynamic memory allocation to acquire storage space on the heap



Dynamic Memory Allocation

- Use dynamic memory allocation to acquire storage space on the heap
  - Variables on the stack have a scope based on the function they are declared in
  - Heap memory is allocated to your program from the time you request the memory until the time you tell the operating system you no longer need it, or until your program ends.
- You can, at runtime, ask for extra storage space, which C++ will give you
- You can use that storage space however you'd like
- You have to explicitly tell the language when you're done using the memory.



Dynamic Memory Allocation: new

- To request memory from the heap to allocate one element:

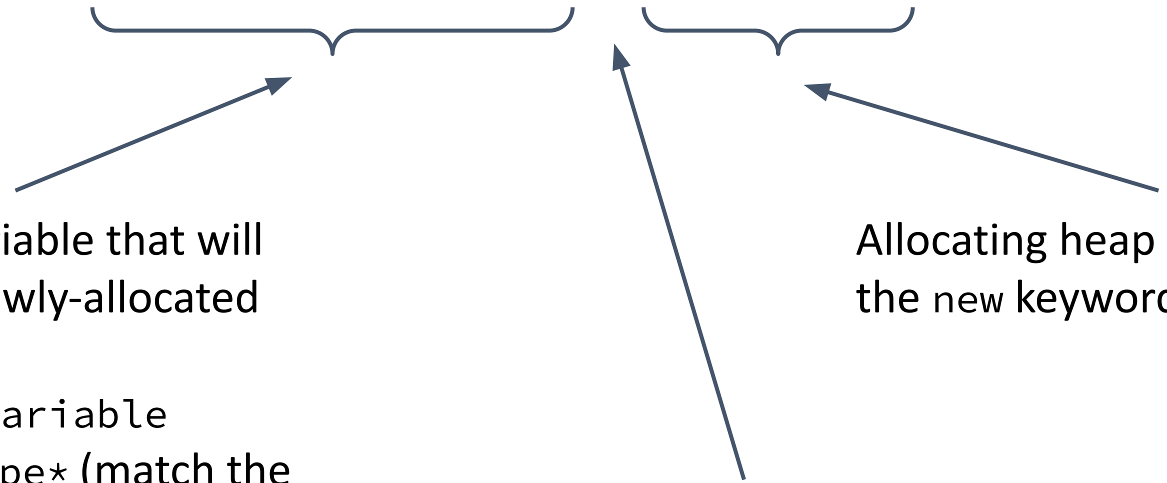
```
type* variable = new type;
```

- To allocate multiple (n) elements on the heap:

```
type* variable = new type[n];
```

Dynamic Memory Allocation: new

```
type* variable = new type;
```



Declaring a variable that will point at our newly-allocated memory

- Name is `variable`
- Type is `type*` (match the type of the element)

Assigning the pointer to point to the heap memory

Allocating heap memory with the `new` keyword

Dynamic Memory Allocation: Examples

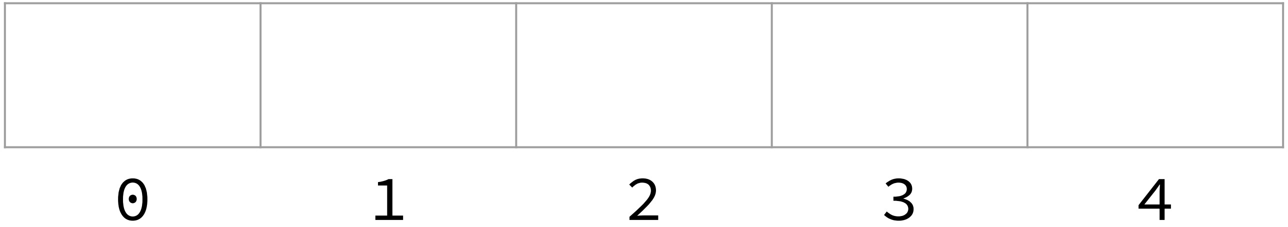
```
int* anInteger = new int;

int* tenInts = new int[10];
```

Pointers

- A pointer is a brand new data type that becomes very prominent when working with dynamically allocated memory.
- Just like all other data types, pointers take up space in memory and can store specific values.
- A pointer always stores a memory address, which is like the specific coordinates of where a piece of memory exists on the computer.
- They quite literally "point" to another location on your computer.

Arrays

- Lower-level and more limited than Vectors
- A contiguous chunk of space in the computer's memory, split into slots, each of which can contain one piece of information
  - Contiguous means that each slot is located directly next to the others (There are no "gaps")
  - Have a specific type which dictates what information can be held in each slot
  - Each slot has an "index" by which we can refer to it



Arrays

- `int firstTen[10];`
  - Create an array of 10 ints on the stack
  - Only accessible within the function that created it
- `int* secondTen = new int[10];`
  - Creates an array of 10 ints on the heap
  - Accessible for the rest of the program (if we wish)

Arrays

```
int firstTen[10];

int* secondTen = new int[10];
```

Arrays

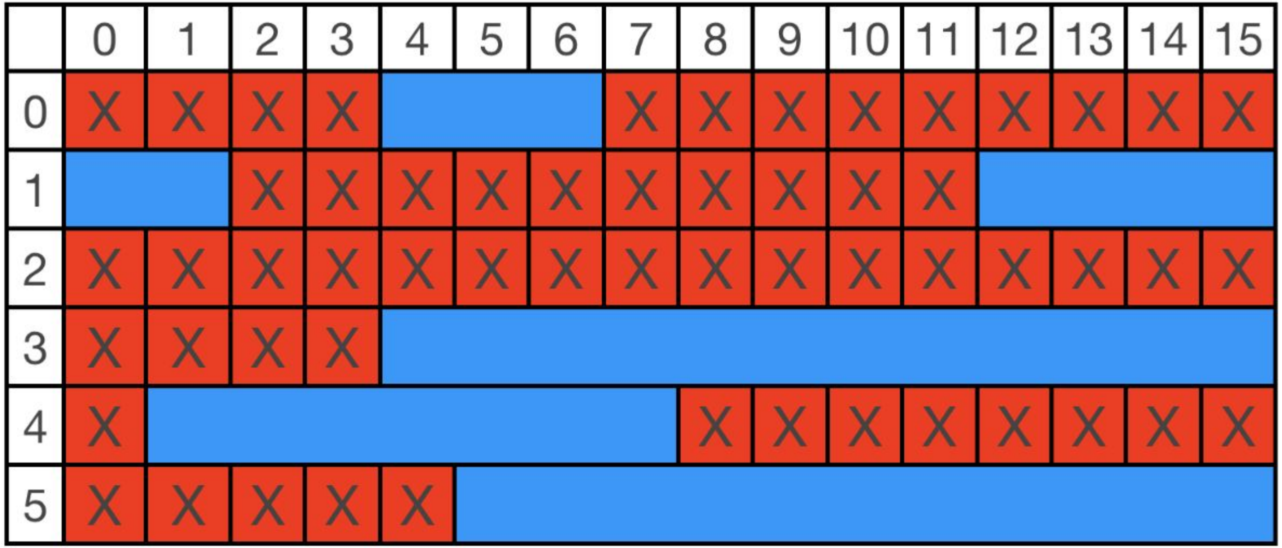
```
int firstTen[10];
int* secondTen = new int[10];
// fill memory with values
for (int i = 0; i < 10; i++) {
    firstTen[i] = i * 2; // evens
    secondTen[i] = i * 2 + 1; // odds
}
```

Arrays

```
int firstTen[10];
int* secondTen = new int[10];
// fill memory with values
for (int i = 0; i < 10; i++) {
    firstTen[i] = i * 2; // evens
    secondTen[i] = i * 2 + 1; // odds
}
int len = firstTen.length(); // ERROR! No functions!
firstTen.add(42); // ERROR! No functions!
```

Under the Hood

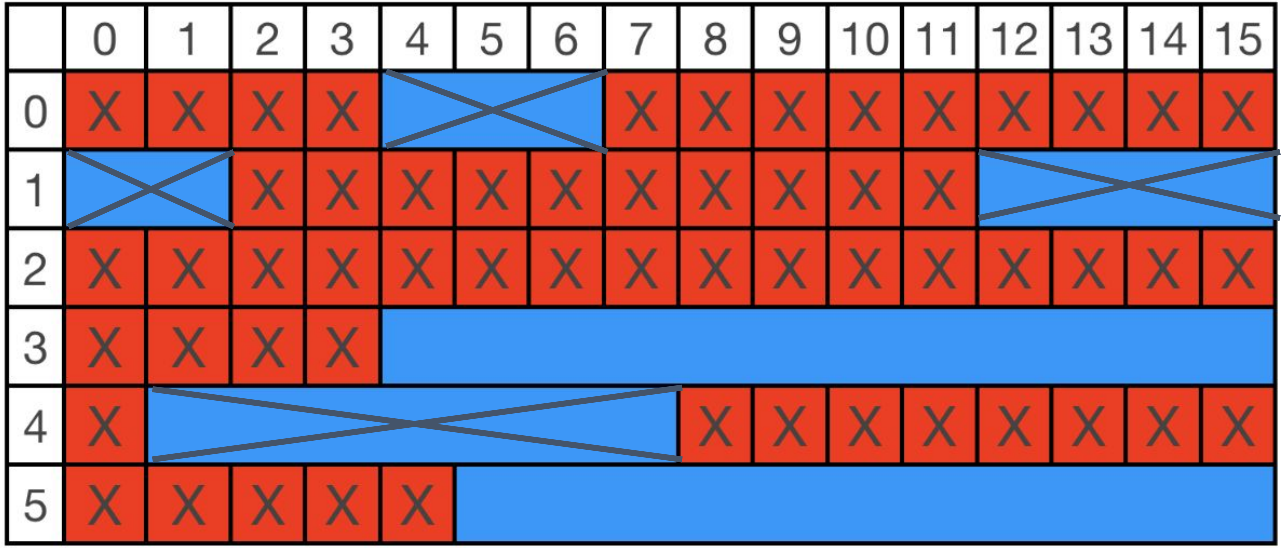
```
int* tenInts = new int[10];
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

Under the Hood

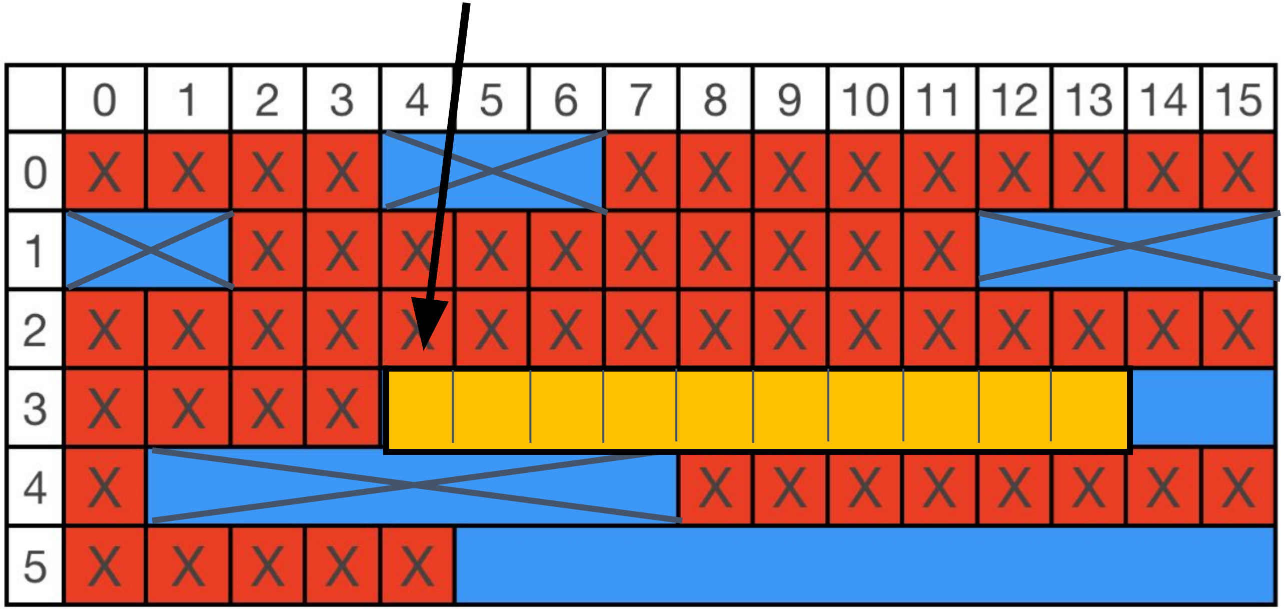
```
int* tenInts = new int[10];
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

Under the Hood

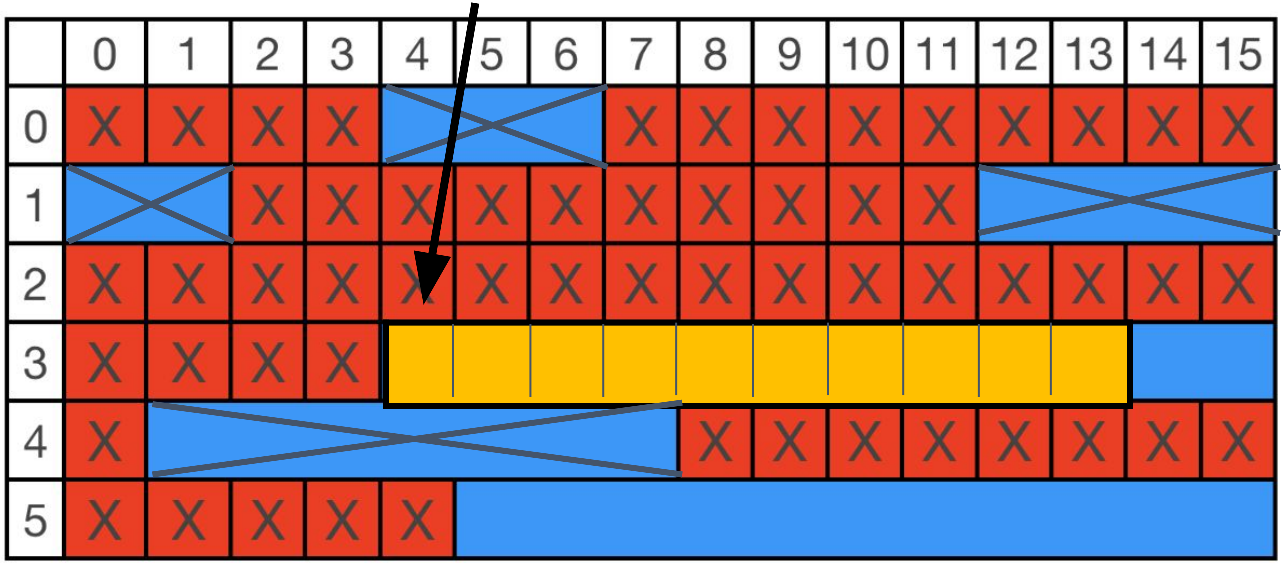
```
int* tenInts = new int[10];
```



Under the Hood

```
int* tenInts = new int[10];
```

(Col 4, Row 3)

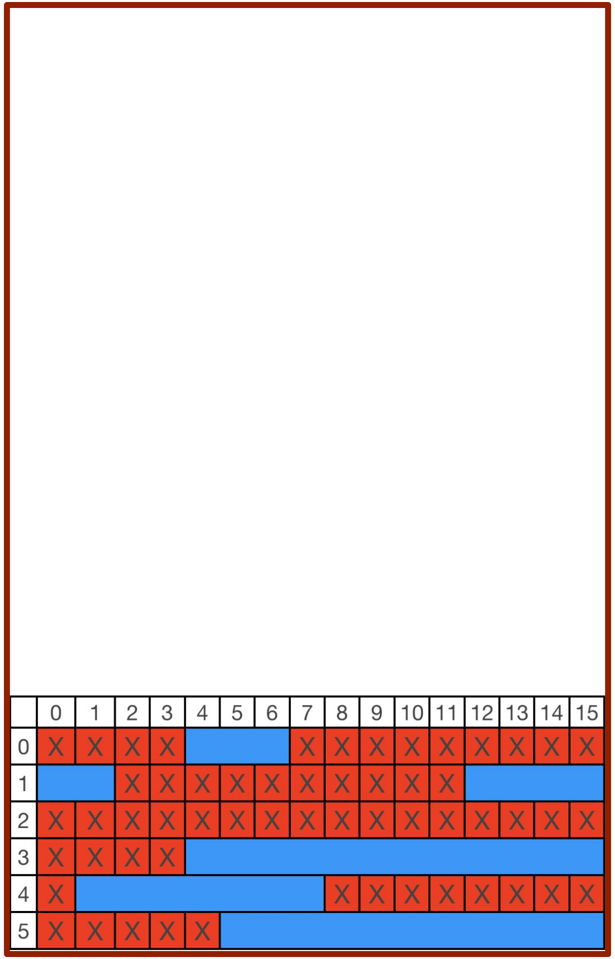


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟦	🟦
4	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

Tracing Example

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```

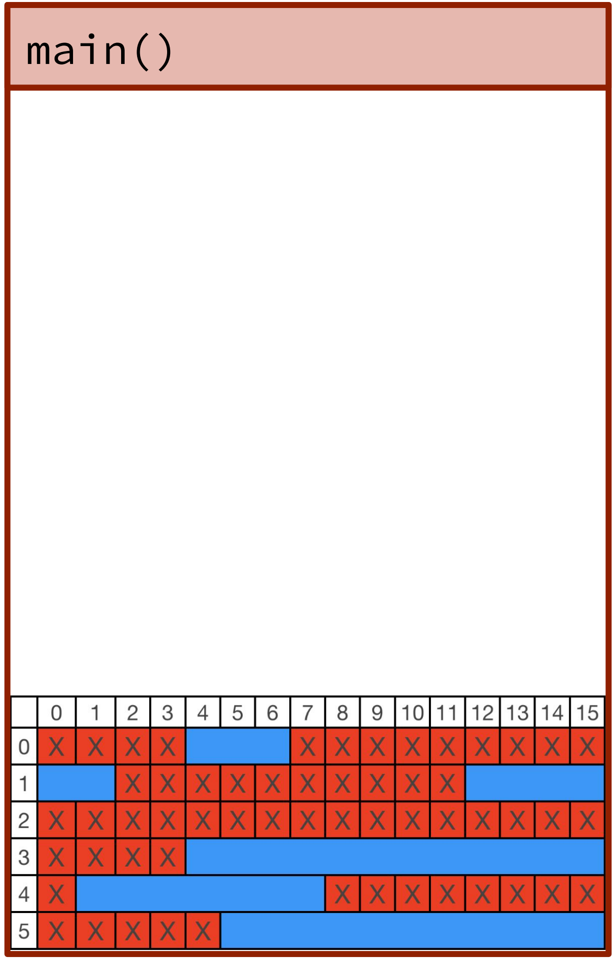


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

```
int main () {
   int numValues = getInteger("How many words?");
   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
     arr[i] = getLine("Enter a string: ");
   }
   for (int i = 0; i < numValues; i++) {
     cout << i << ": " << arr[i] << endl;
   }
   return 0;
}
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

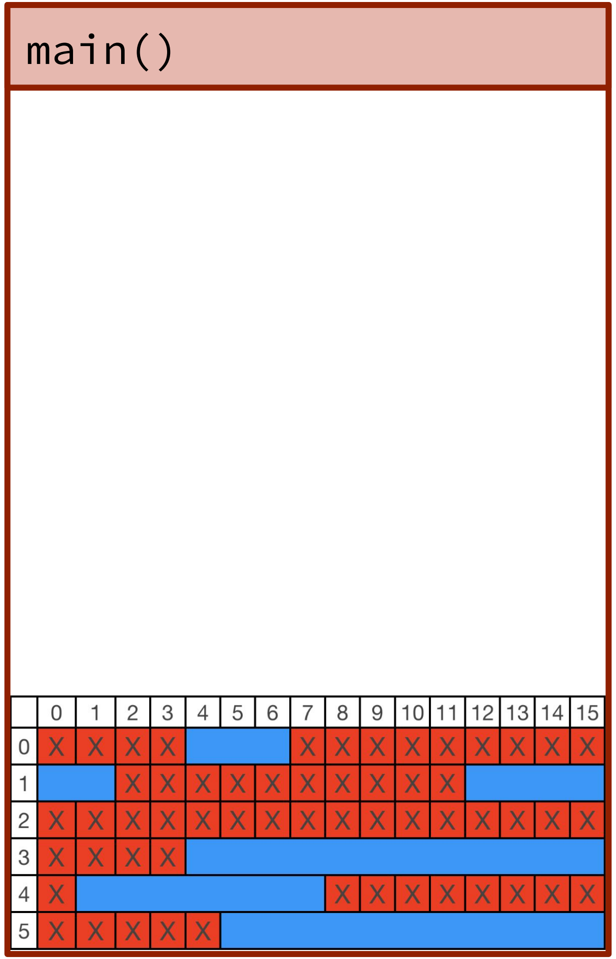
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



main()

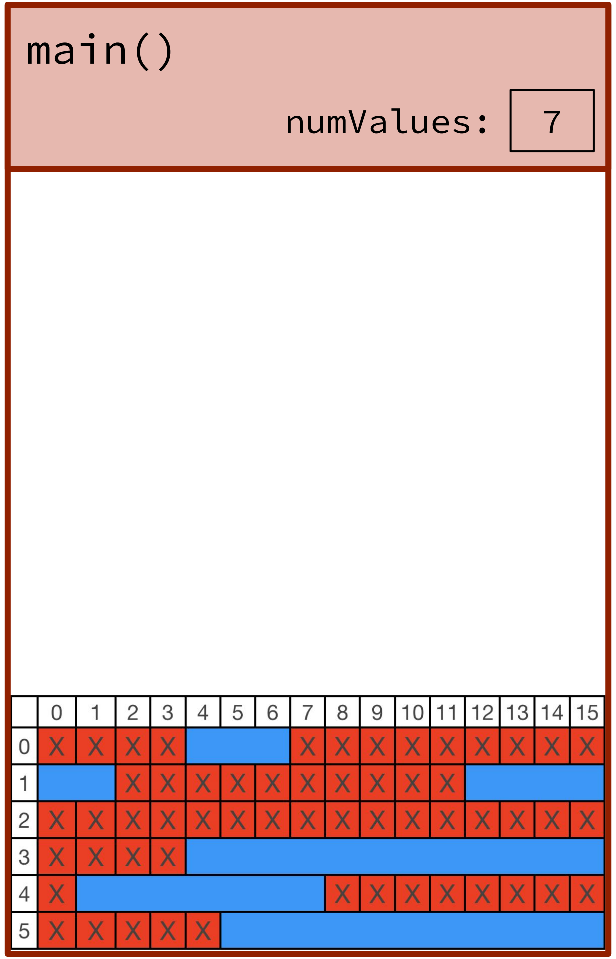
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```

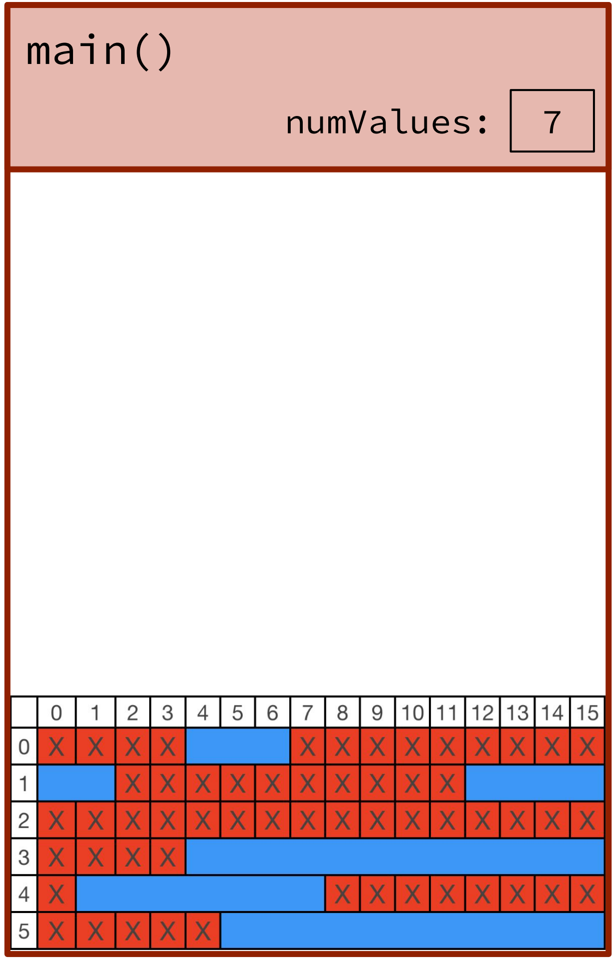


main()

numValues: 7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```

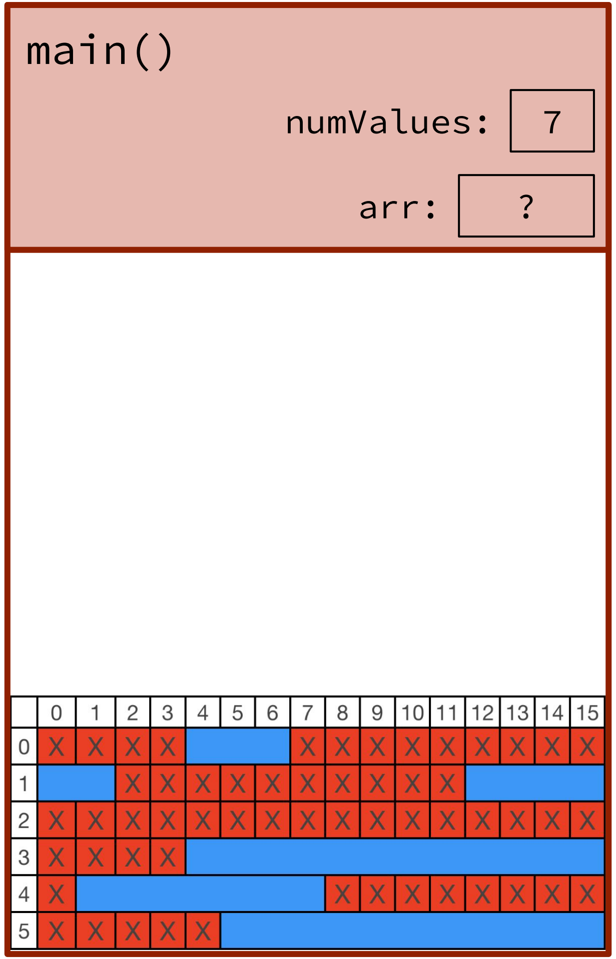


main()

numValues: 7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



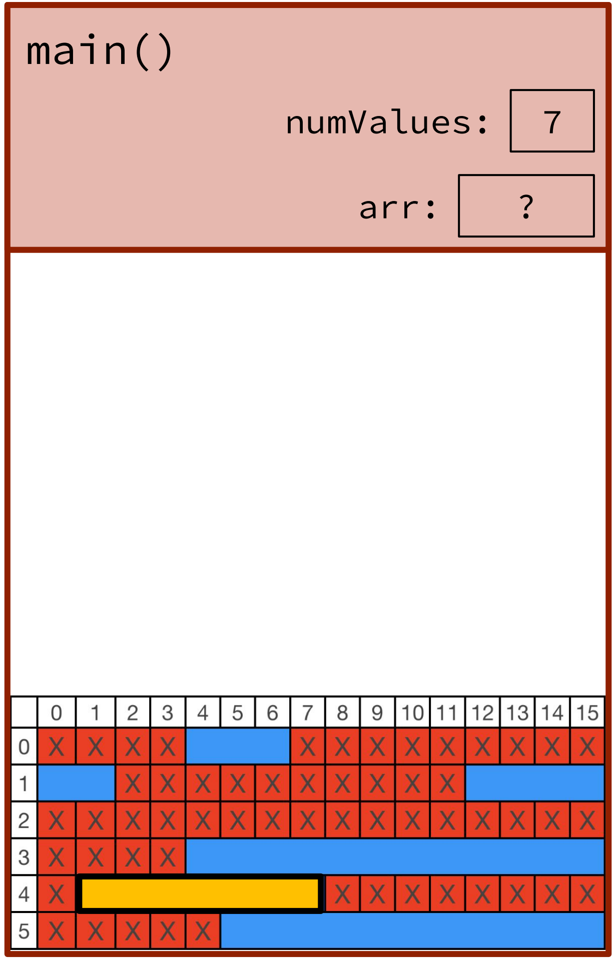
main()

numValues: 7

arr: ?

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



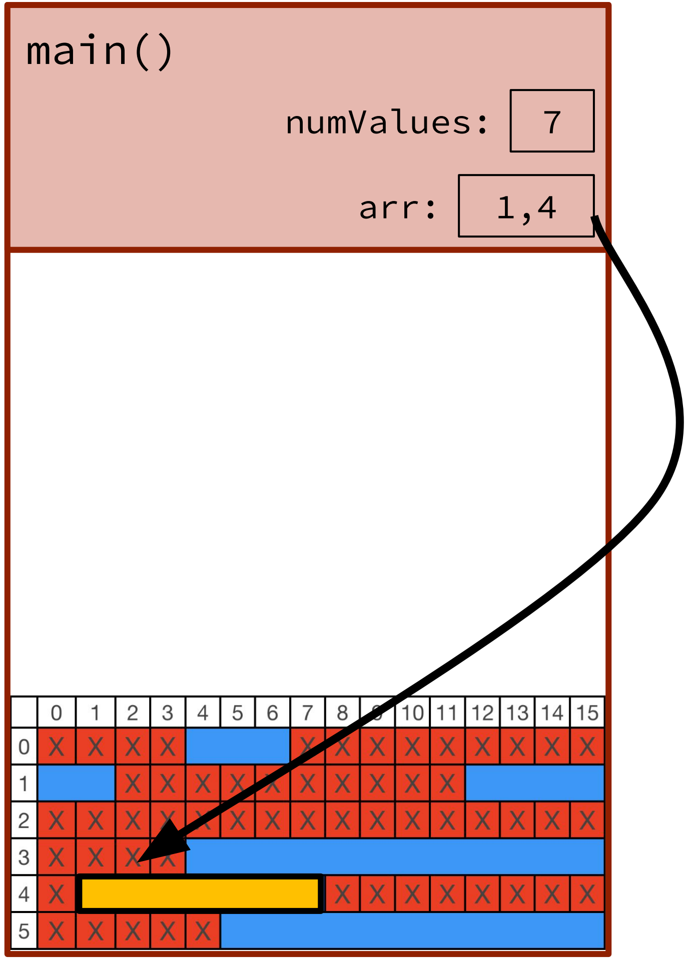
main()

numValues: 7

arr: ?

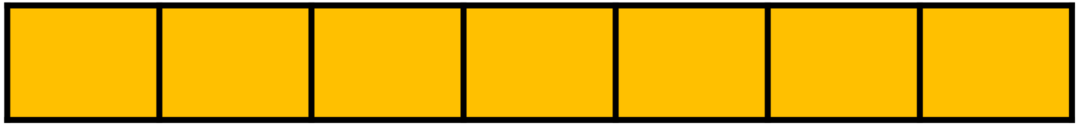
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟨	🟨	🟨	🟨	🟨	🟨	🟨	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

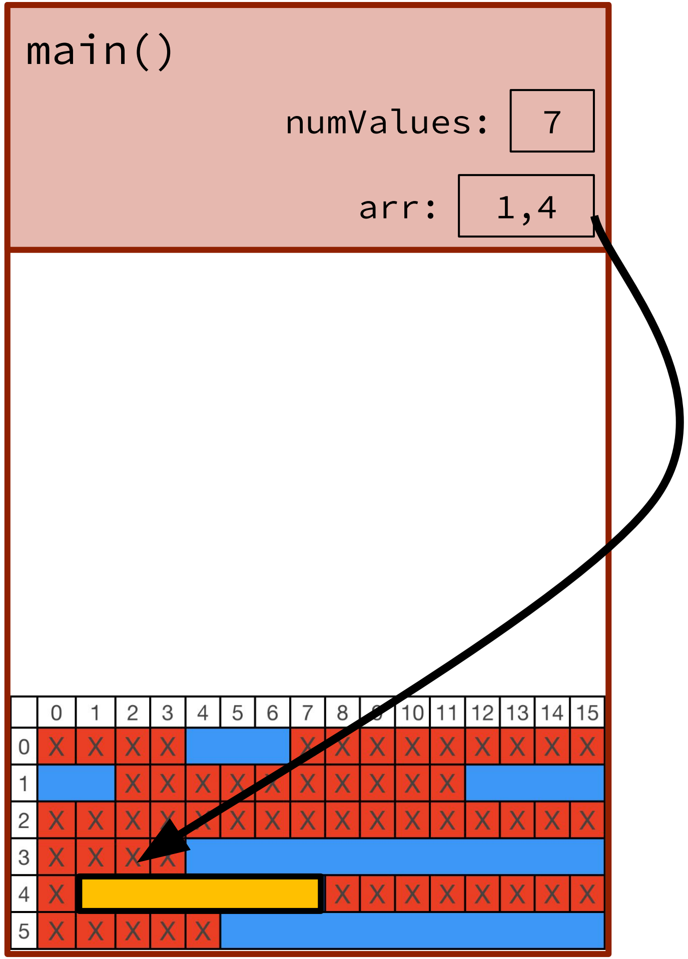
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟨	🟨	🟨	🟨	🟨	🟨	🟨	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```





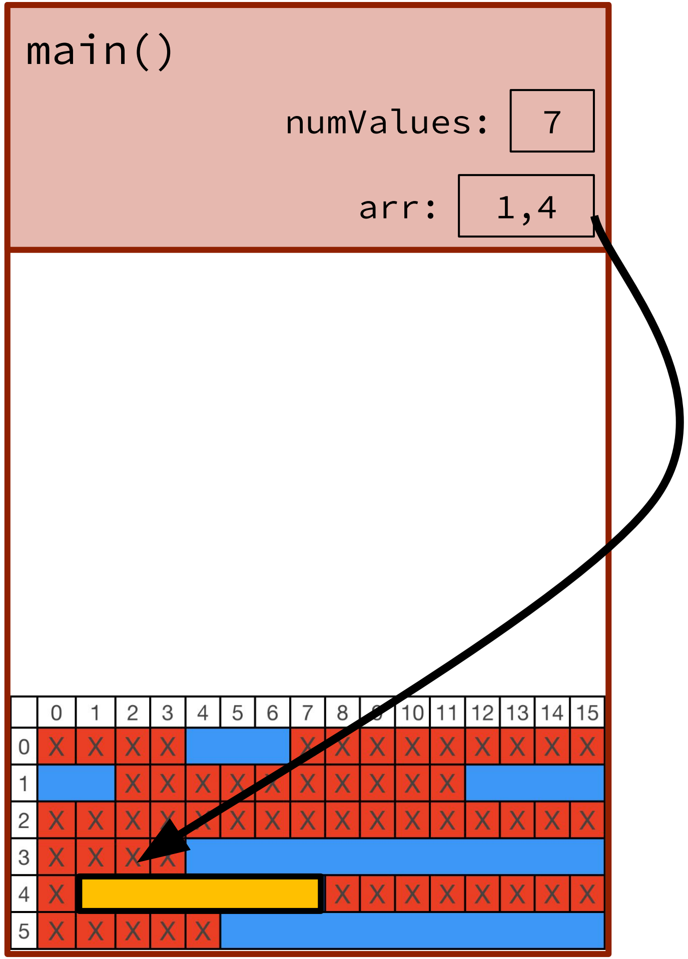
main()

numValues: 7

arr: 1,4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```

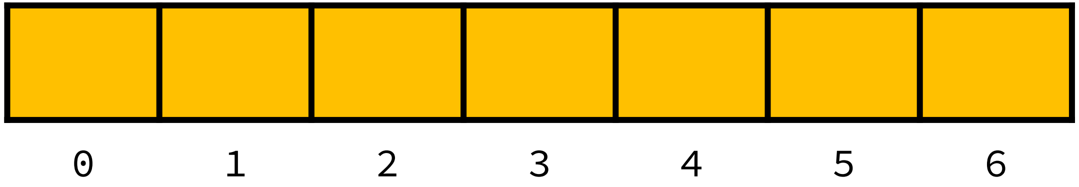


main()

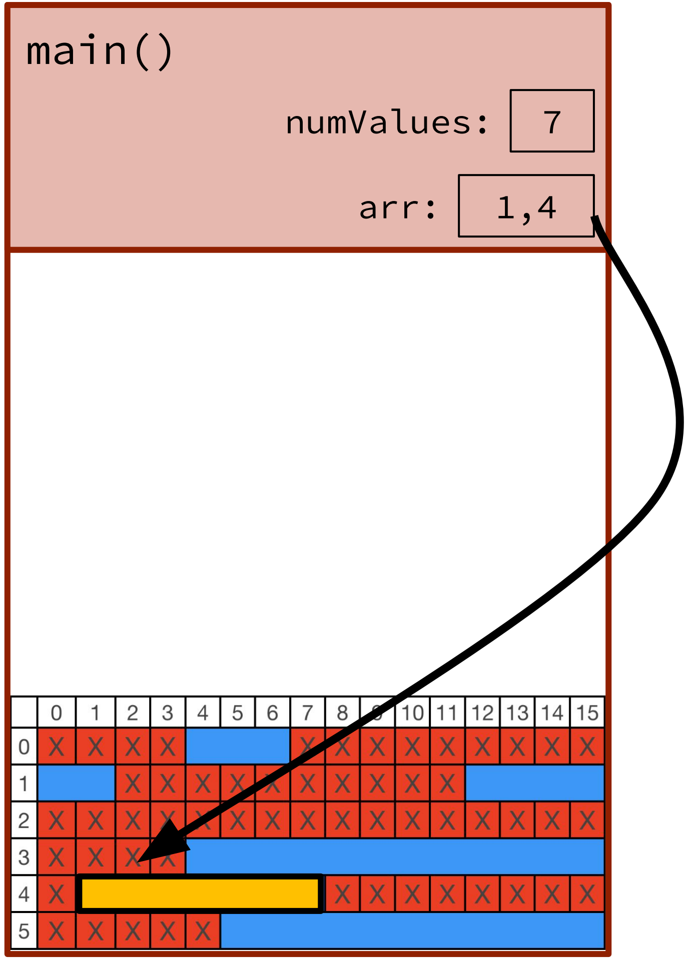
numValues: 7

arr: 1,4

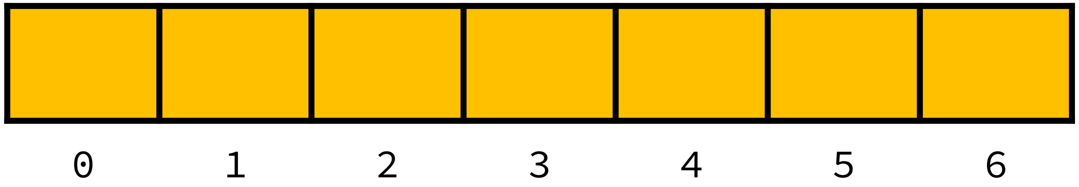
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟨	🟨	🟨	🟨	🟨	🟨	🟨	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦



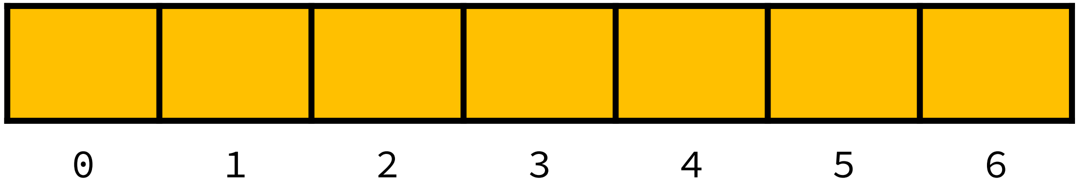
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```

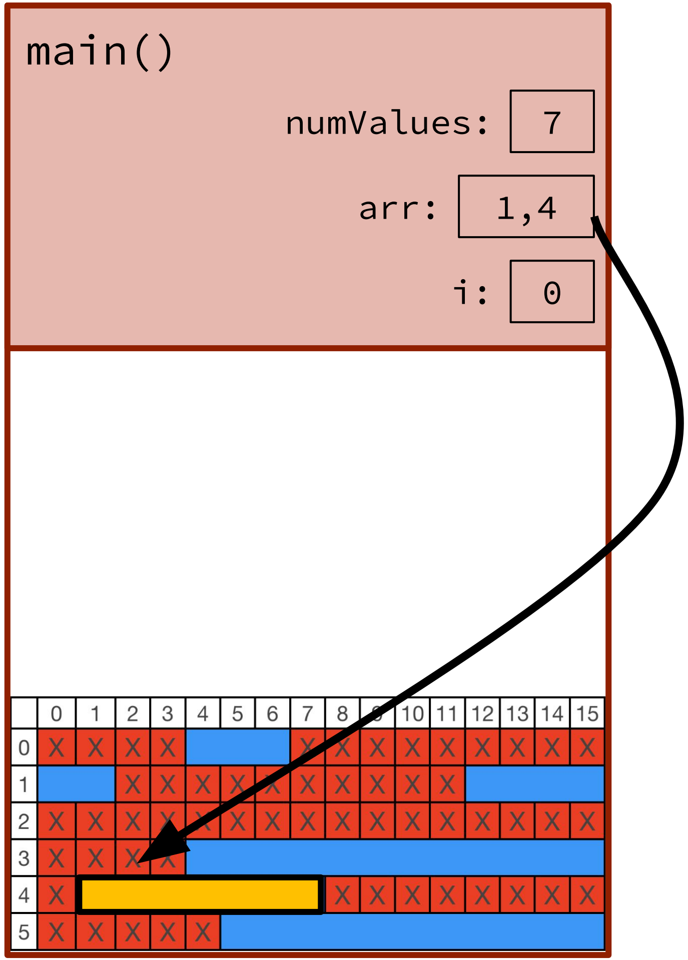


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟨	🟨	🟨	🟨	🟨	🟨	🟨	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦



```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```

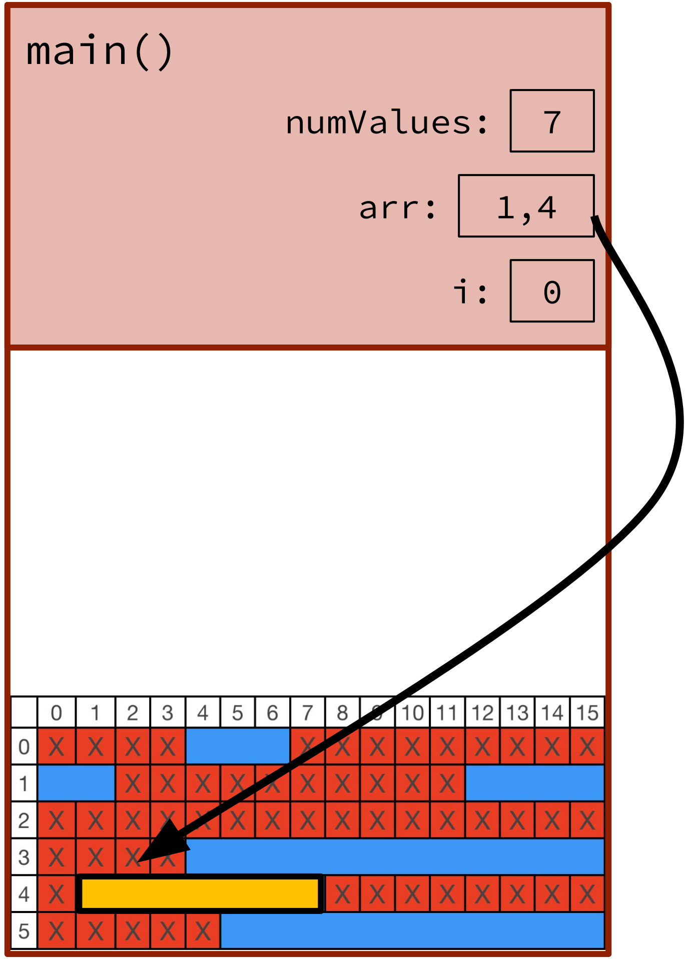




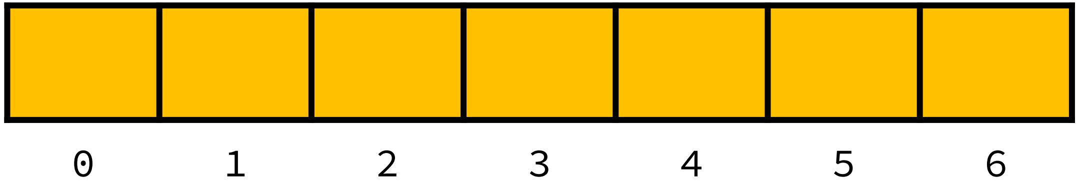
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

Stanford University

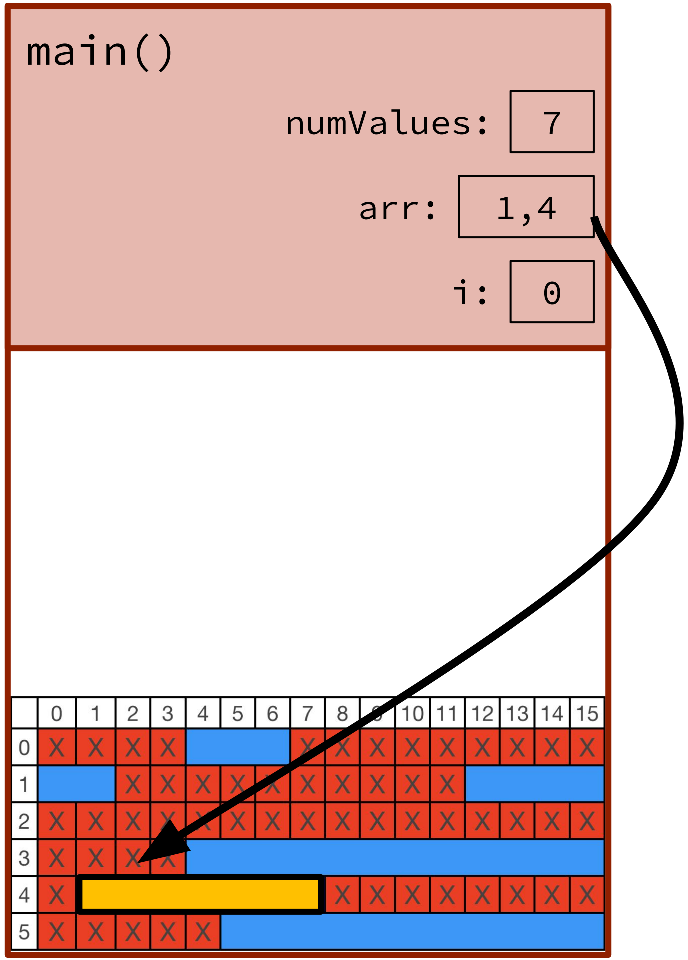
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											



```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



main()

numValues: 7

arr: 1,4

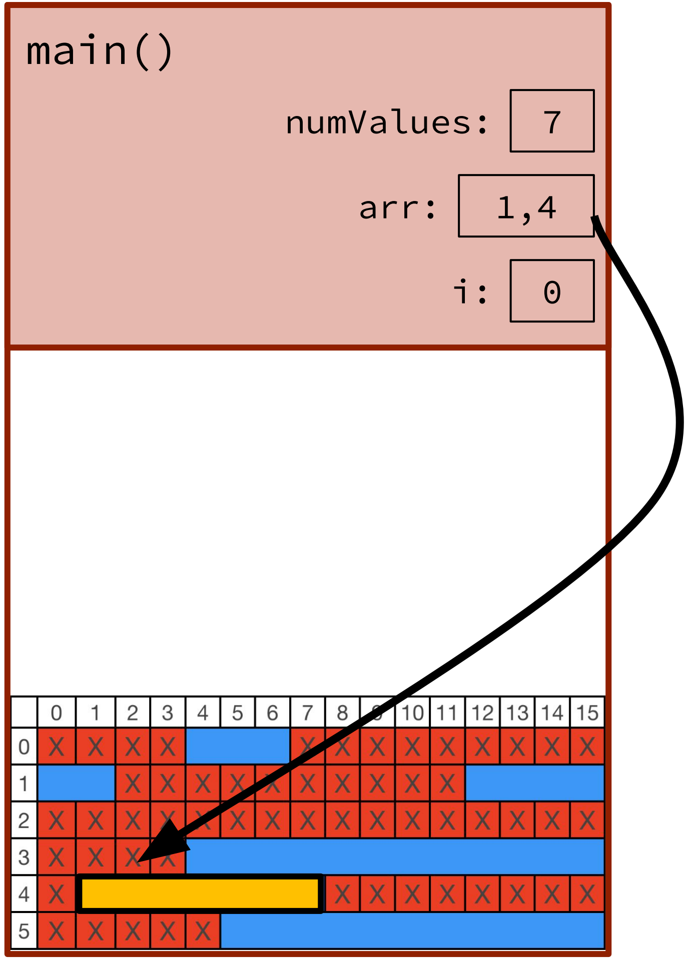
i: 0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟨	🟨	🟨	🟨	🟨	🟨	🟨	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

Watch						
0	1	2	3	4	5	6

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```

Watch						
0	1	2	3	4	5	6



main()

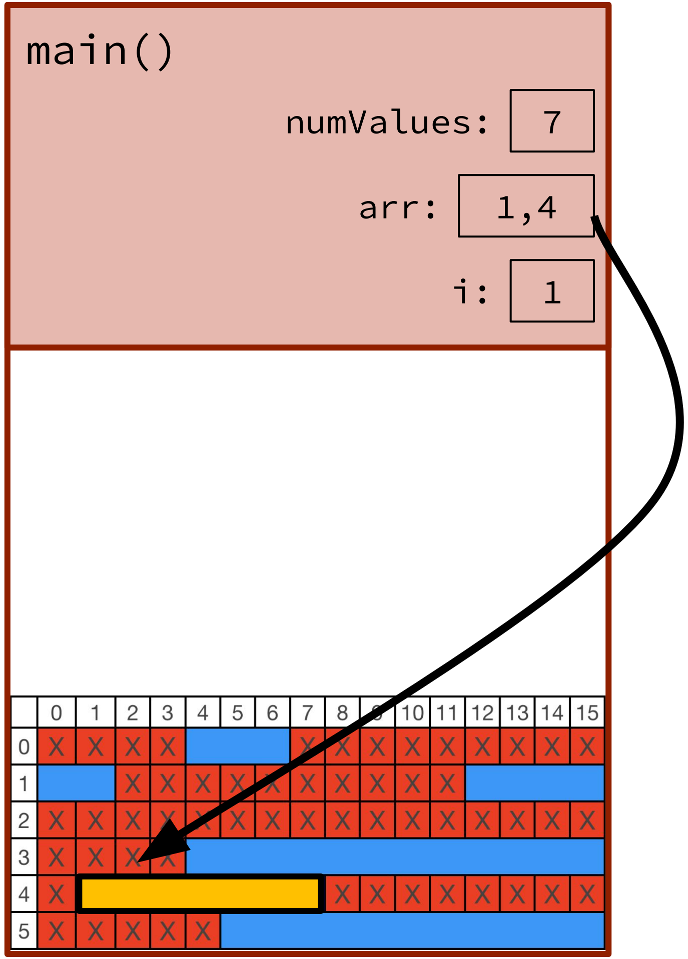
numValues: 7

arr: 1,4

i: 0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```

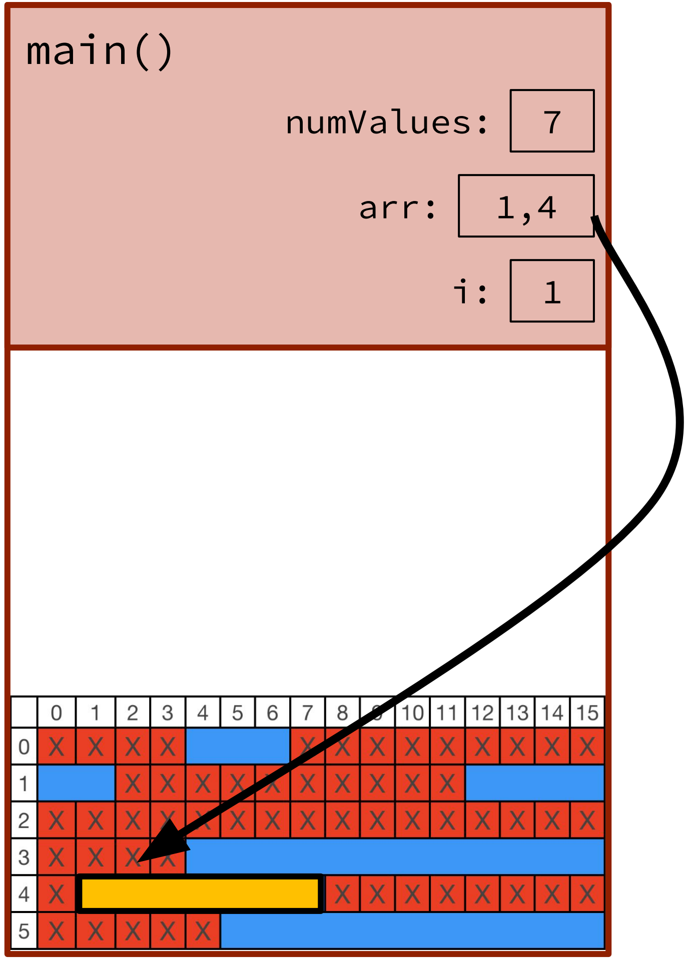


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟨	🟨	🟨	🟨	🟨	🟨	🟨	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

Watch						
0	1	2	3	4	5	6

Stanford University

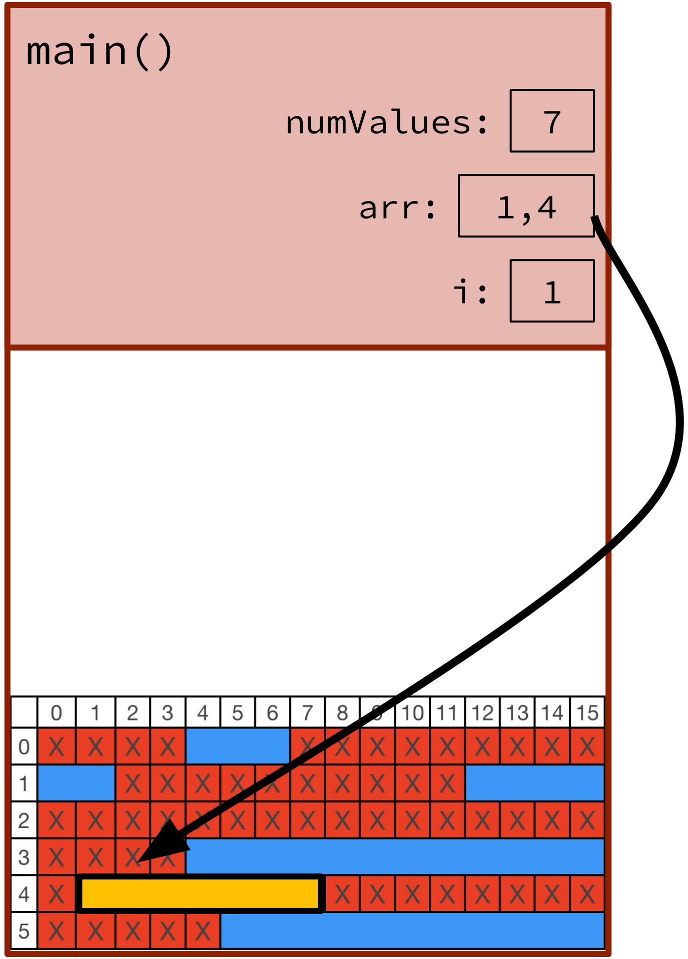
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

Watch	me					
0	1	2	3	4	5	6

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



main()

numValues: 7

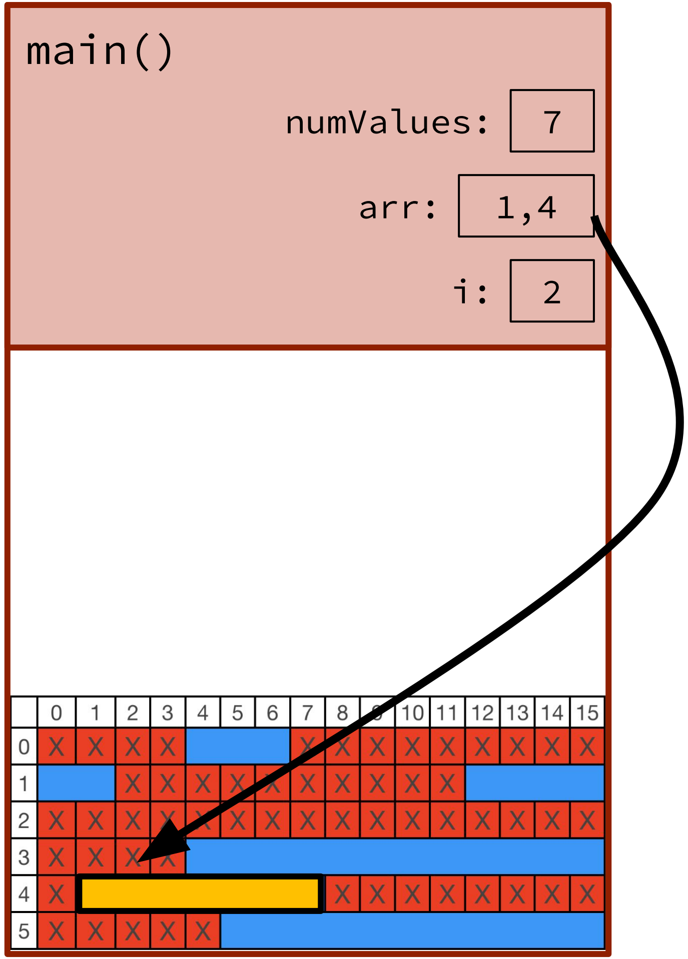
arr: 1,4

i: 1

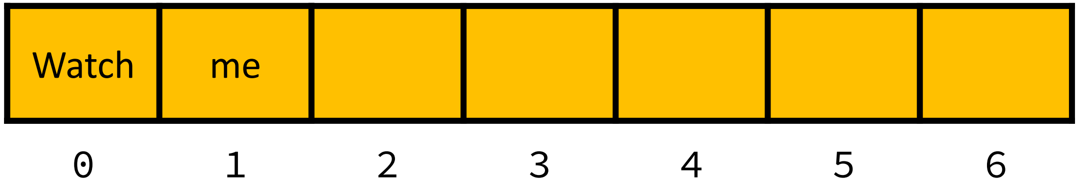
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

Watch	me					
0	1	2	3	4	5	6

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```

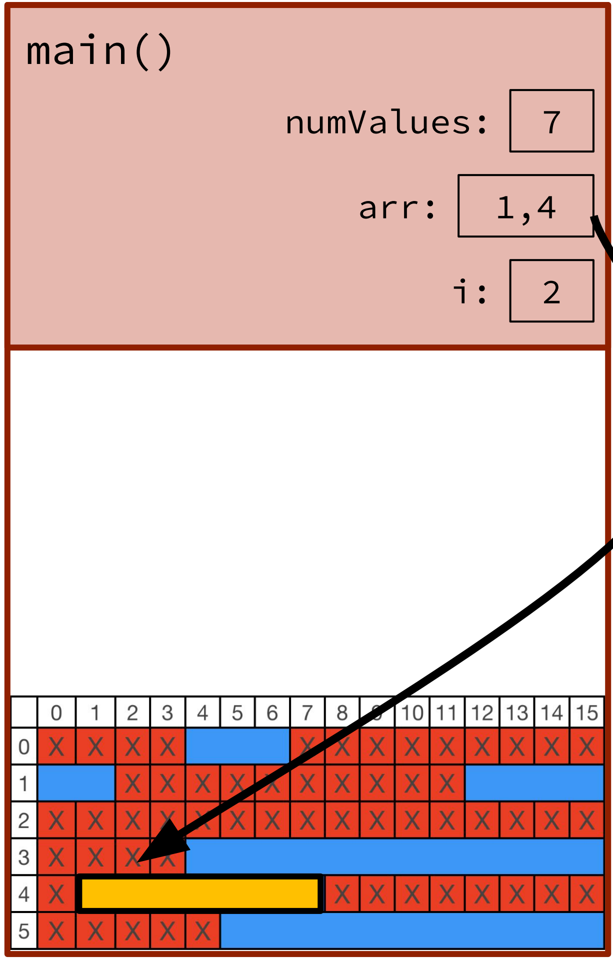


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											



0	1	2	3	4	5	6
Watch	me					

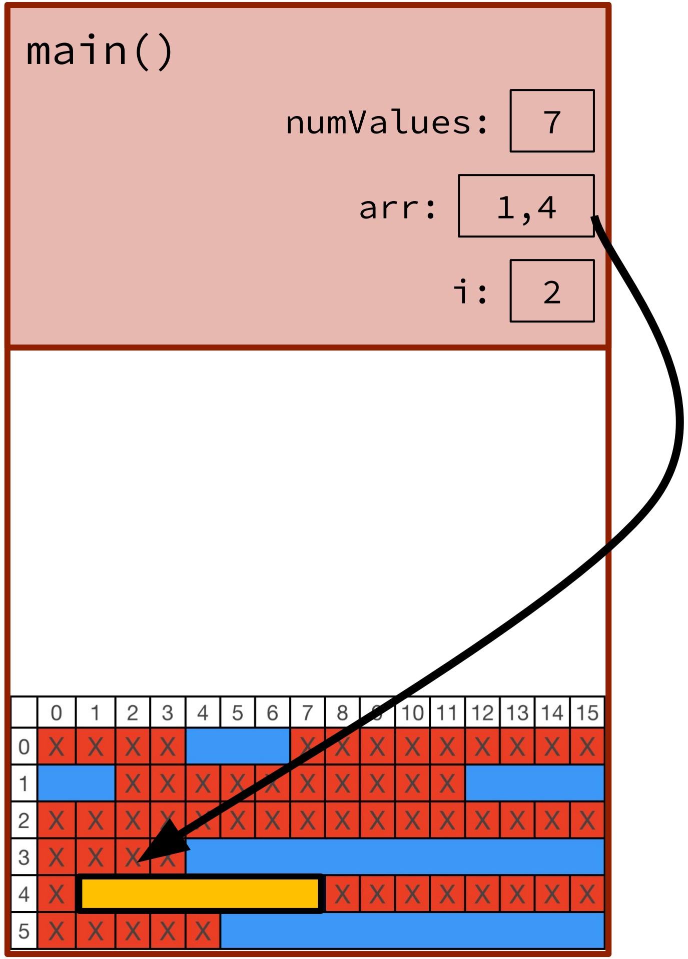
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

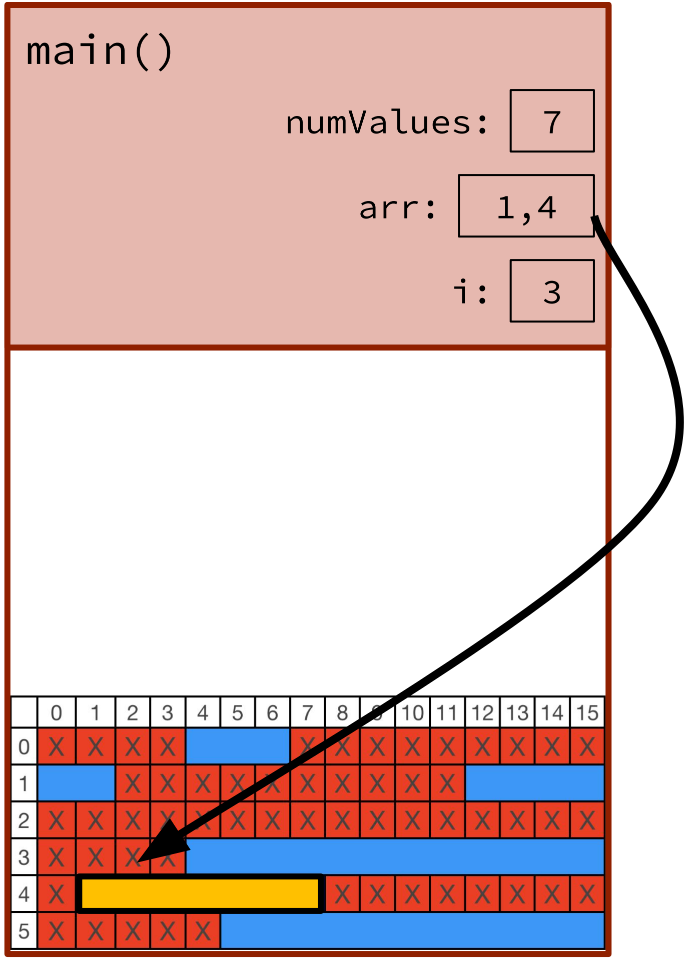
Watch	me	dance,				
0	1	2	3	4	5	6

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



Watch	me	dance,				
0	1	2	3	4	5	6

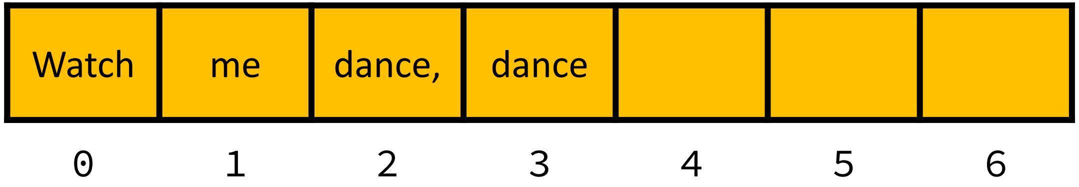
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



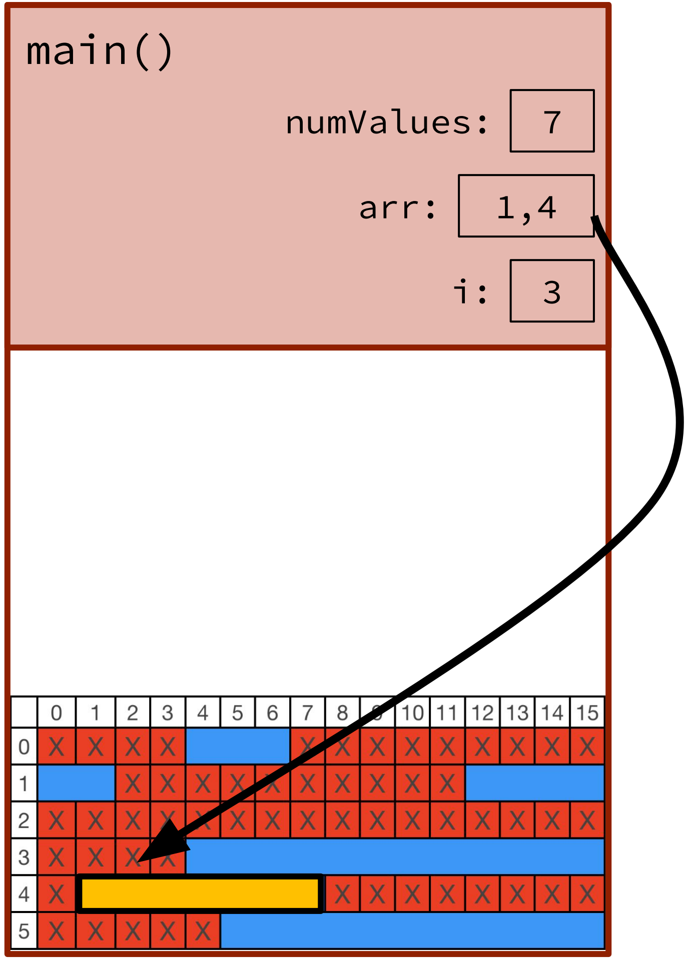
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

Watch	me	dance,				
0	1	2	3	4	5	6

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



Watch	me	dance,	dance			
0	1	2	3	4	5	6



main()

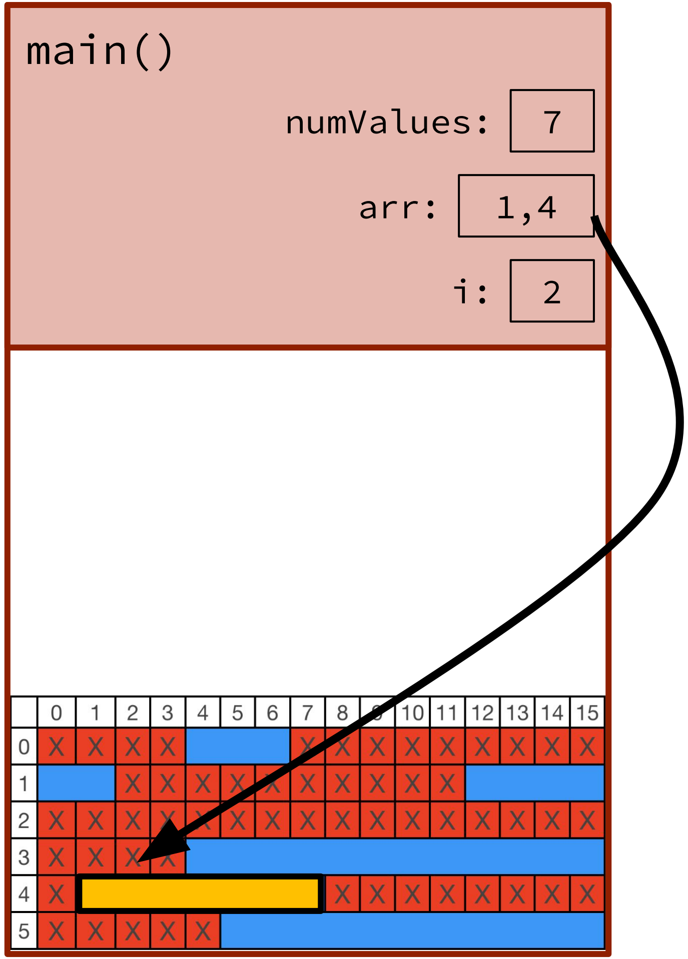
numValues: 7

arr: 1,4

i: 3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

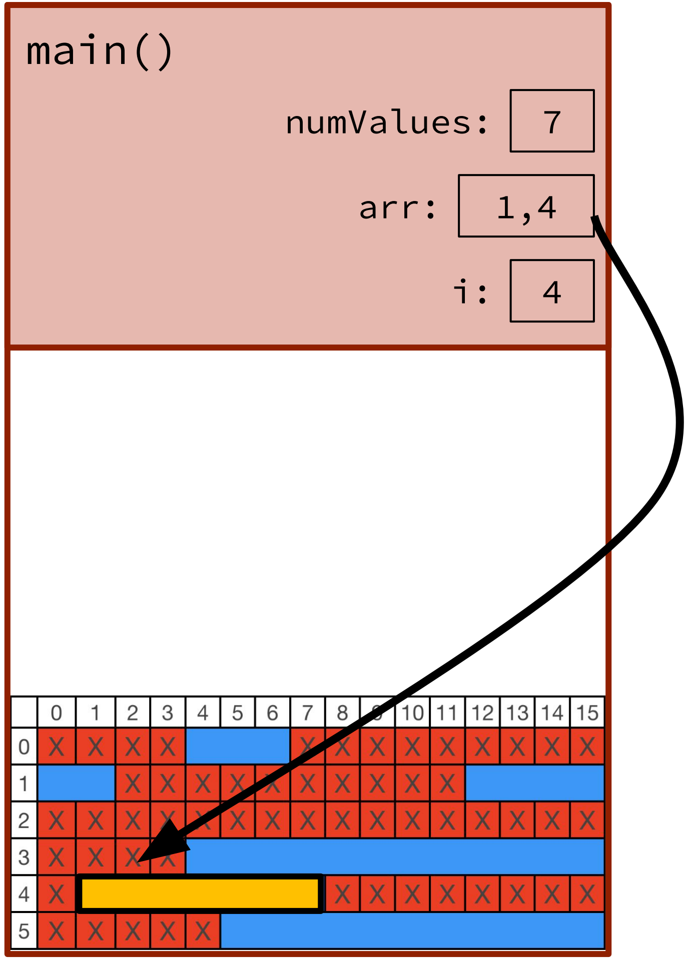
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟨	🟨	🟨	🟨	🟨	🟨	🟨	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

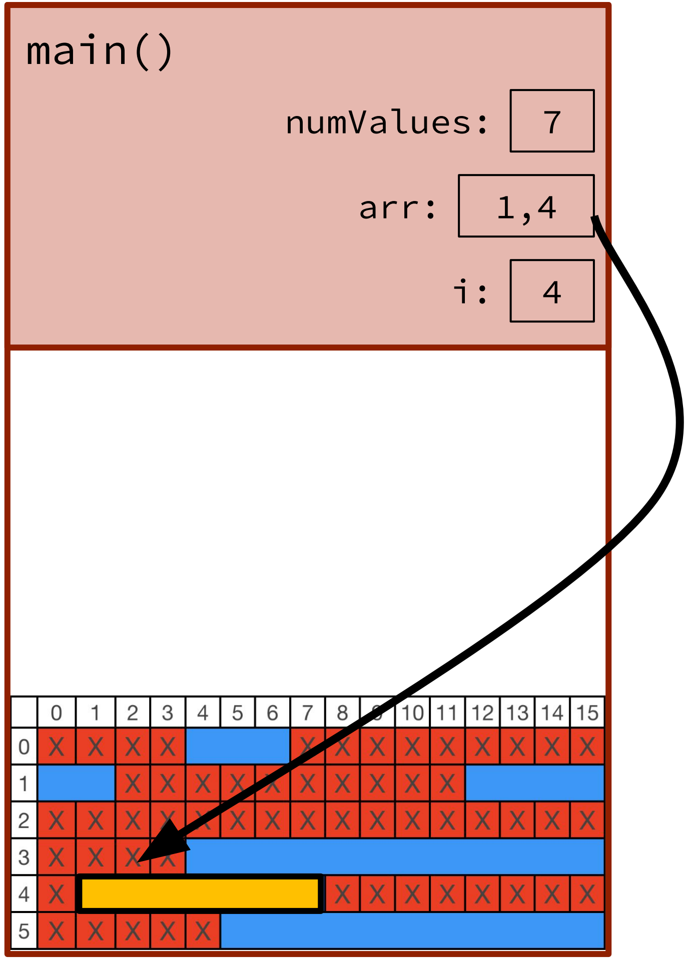
Watch	me	dance,	dance			
0	1	2	3	4	5	6

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



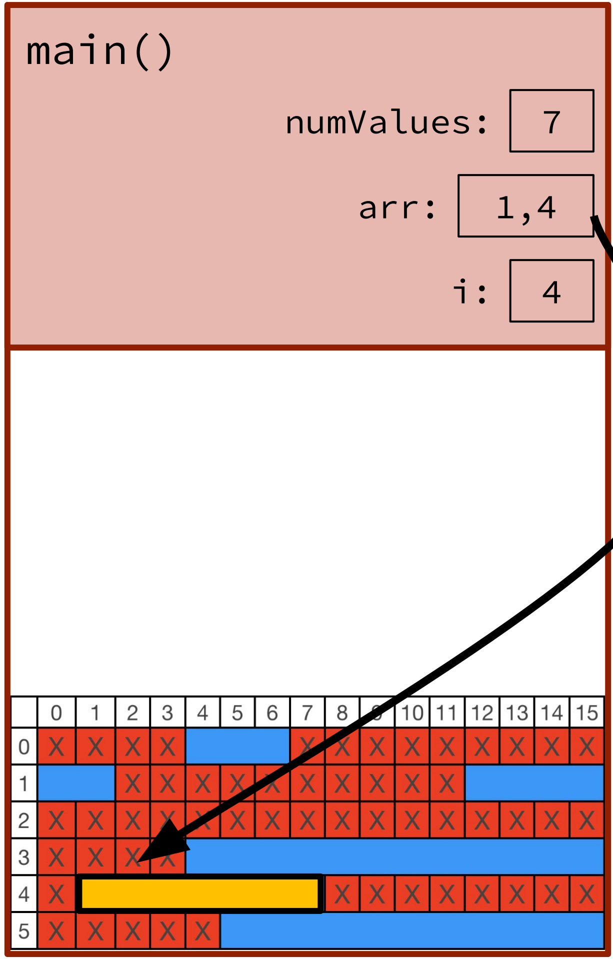
Watch	me	dance,	dance			
0	1	2	3	4	5	6

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



Watch	me	dance,	dance	the		
0	1	2	3	4	5	6

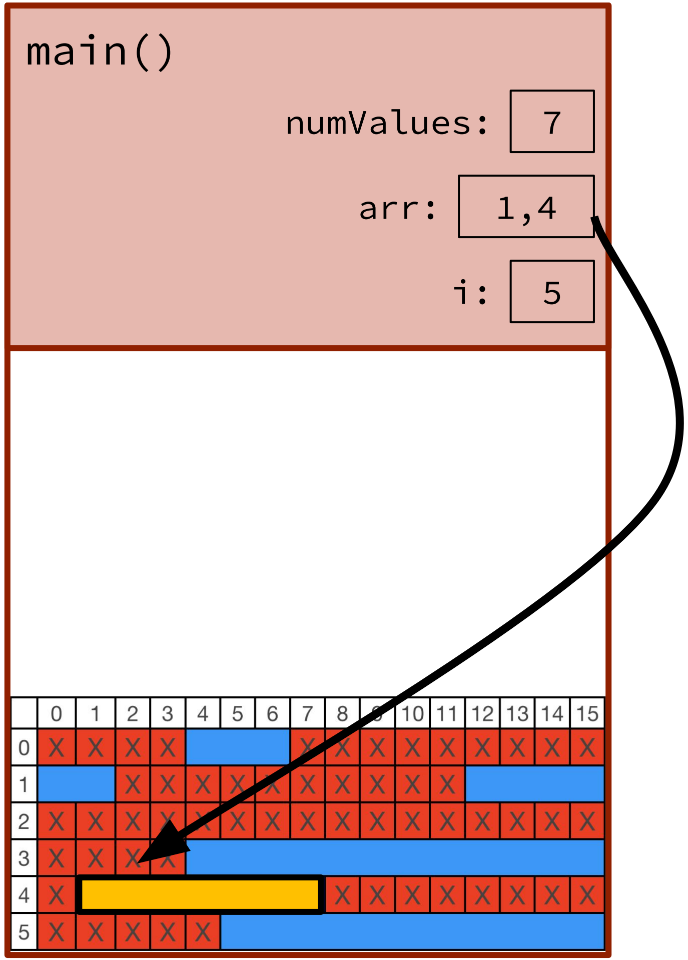
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

Watch	me	dance,	dance	the		
0	1	2	3	4	5	6

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



main()

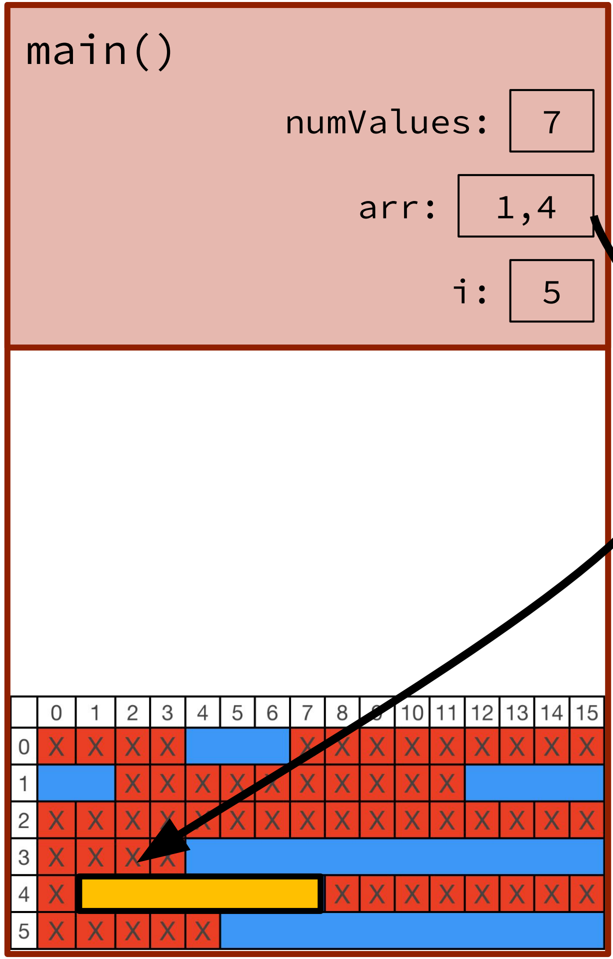
numValues: 7

arr: 1,4

i: 5

Watch	me	dance,	dance	the		
0	1	2	3	4	5	6

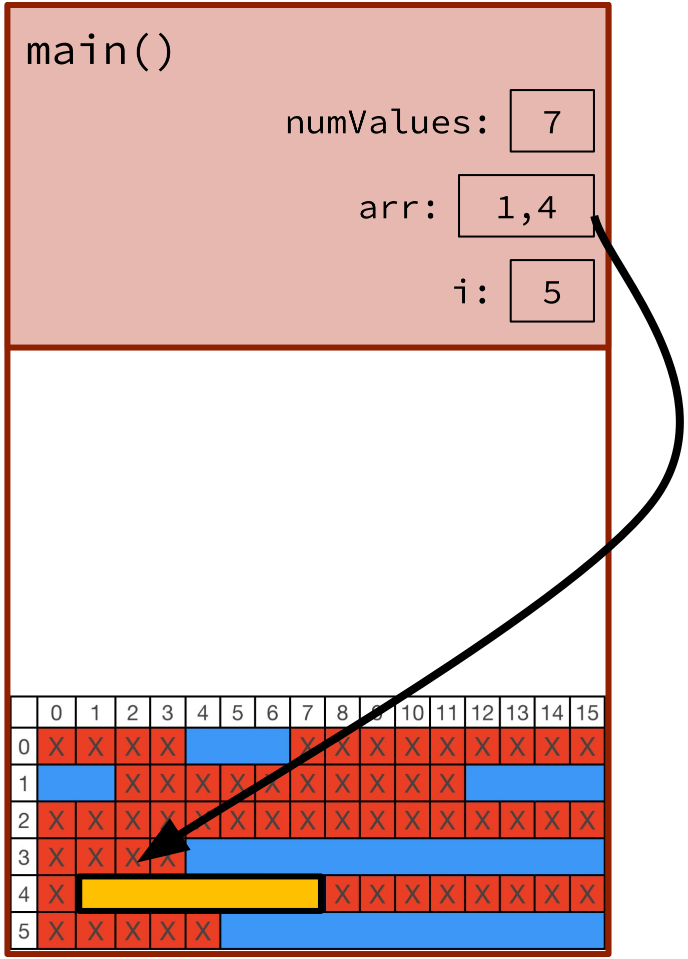
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

Watch	me	dance,	dance	the	night	
0	1	2	3	4	5	6

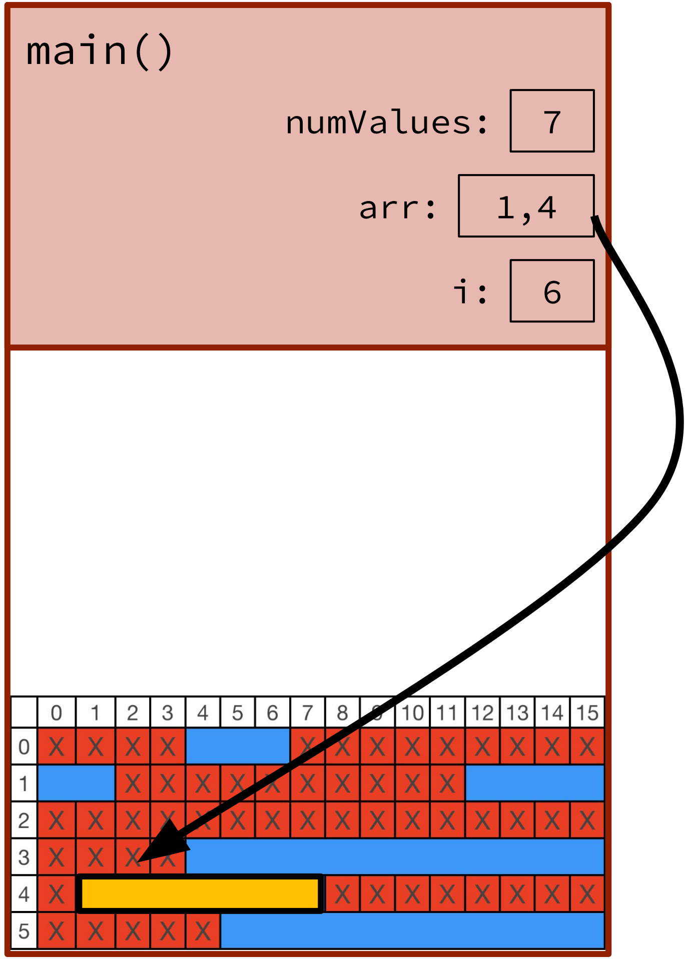
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟨	🟨	🟨	🟨	🟨	🟨	🟨	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

Watch	me	dance,	dance	the	night	
0	1	2	3	4	5	6

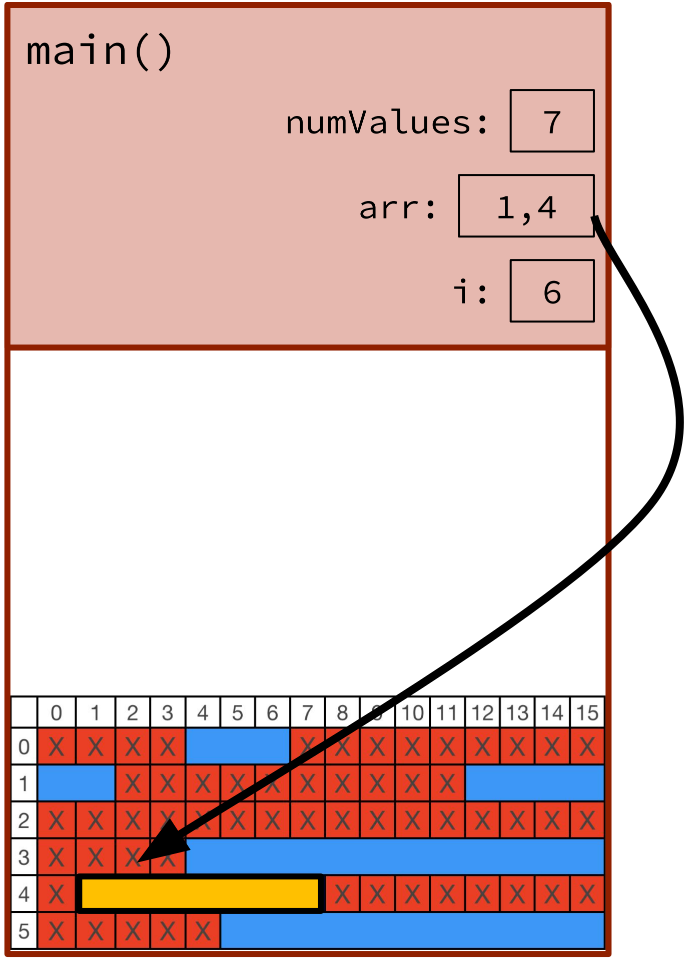
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

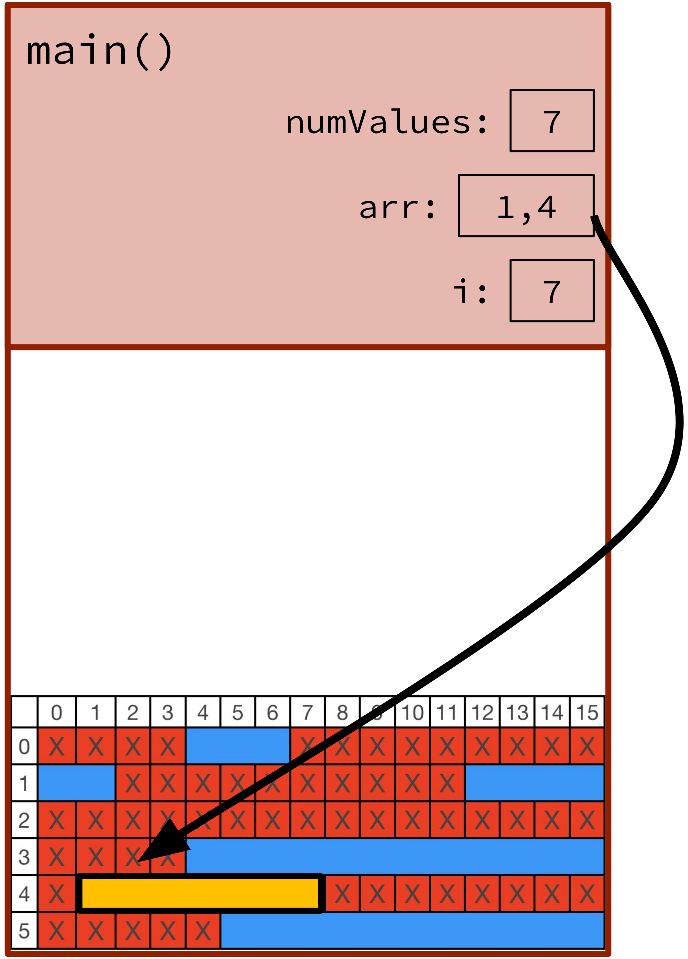
Watch	me	dance,	dance	the	night	
0	1	2	3	4	5	6

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



Watch	me	dance,	dance	the	night	away!
0	1	2	3	4	5	6

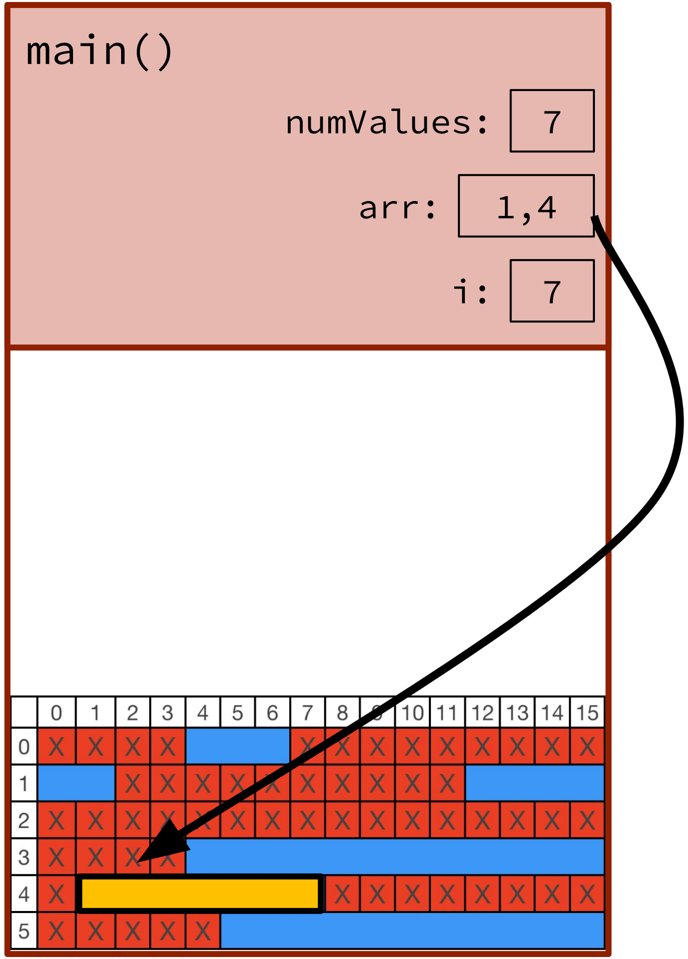
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟨	🟨	🟨	🟨	🟨	🟨	🟨	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

Watch	me	dance,	dance	the	night	away!
0	1	2	3	4	5	6

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



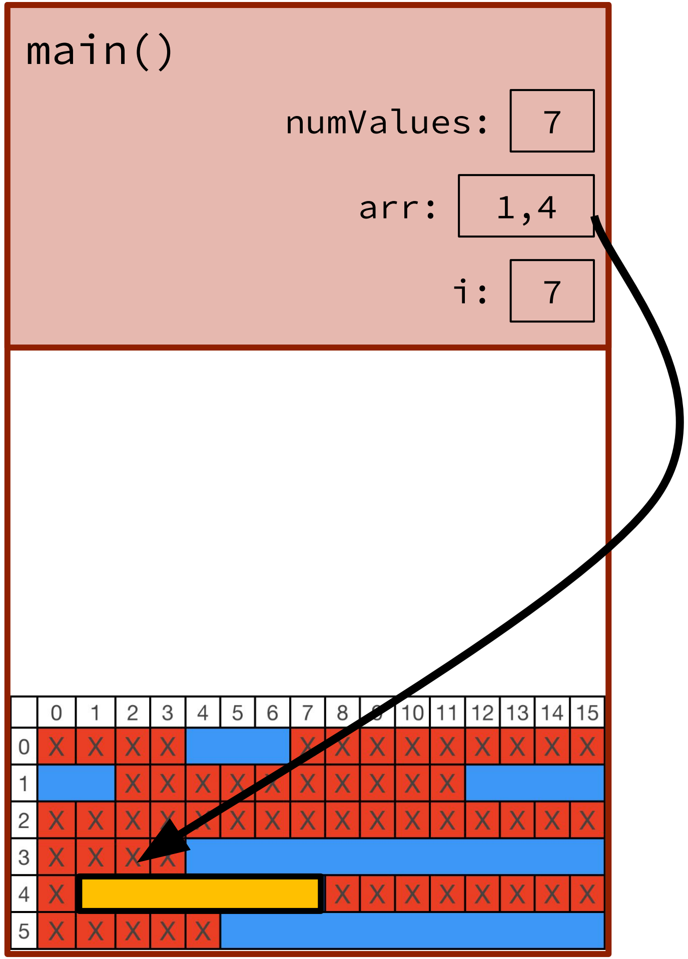
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟨	🟨	🟨	🟨	🟨	🟨	X	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

Watch	me	dance,	dance	the	night	away!
0	1	2	3	4	5	6

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```

```
0: Watch
1: me
2: dance,
3: dance
4: the
5: night
6: away!
```

Watch	me	dance,	dance	the	night	away!
0	1	2	3	4	5	6



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

Pitfalls and Dangers

- C++'s language philosophy prioritizes speed over safety and simplicity
- The array you get from `new[]` is **fixed-size**: it can neither grow nor shrink once it's created
    - C++ does not make that size available to the programmer
    - So, programs that work with arrays typically need an additional variable to keep track of the number of elements
- The array you get from `new[]` has **no bounds-checking**: accessing anything past the beginning or end of an array triggers undefined behavior

Attendance Ticket

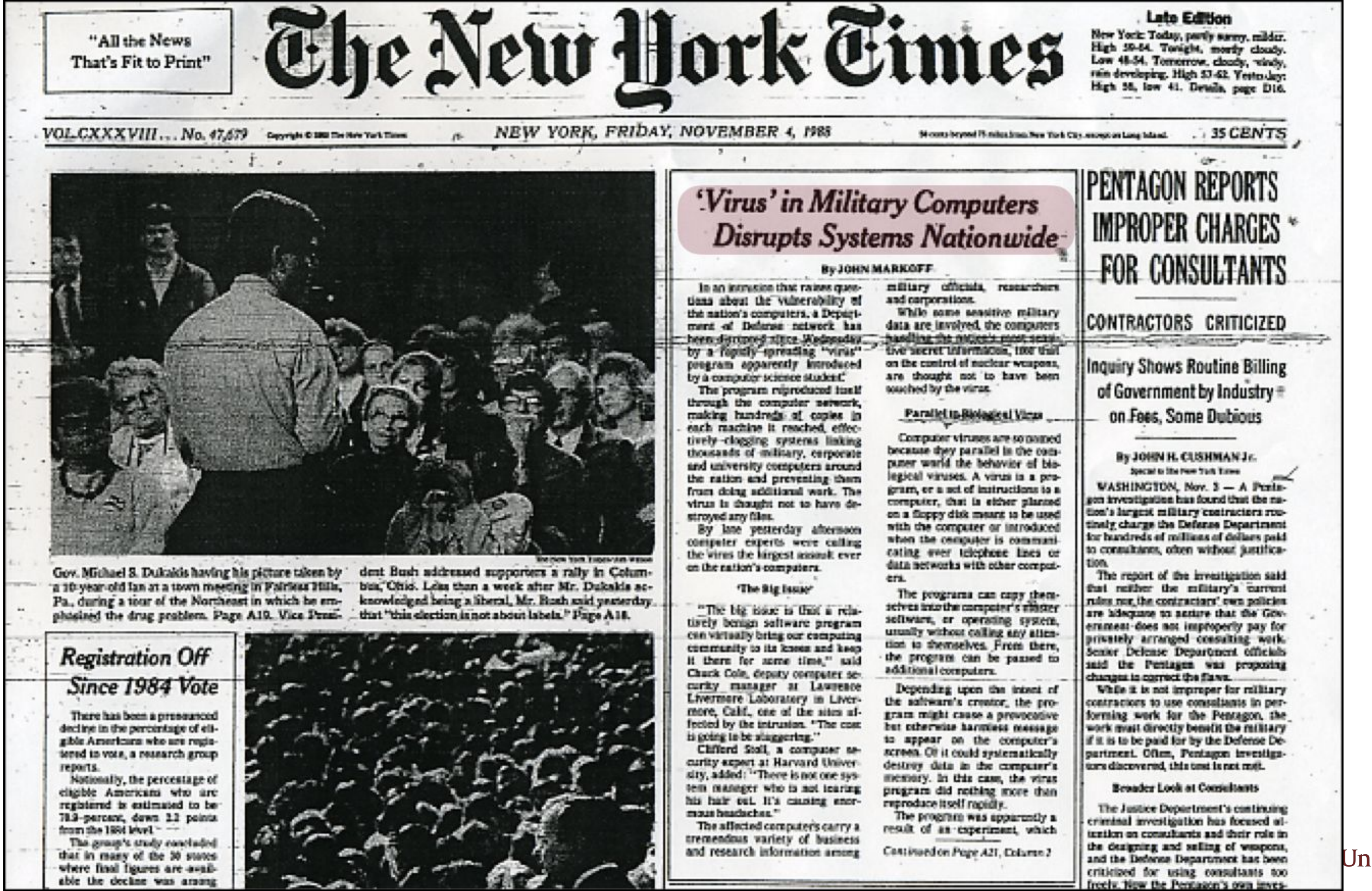
What are potential examples of "undefined behavior" that could occur if you access beyond the bounds of an array? Select all that apply.

- Nothing happens.
- You get a random, garbage value back.
- Your program crashes.
- You make your computer vulnerable to a hacker.

Attendance Ticket

What are potential examples of "undefined behavior" that could occur if you access beyond the bounds of an array? Select all that apply.

- Nothing happens.
- You get a random, garbage value back.
- Your program crashes.
- You make your computer vulnerable to a hacker.
- You make the front page of the New York Times!



"All the News That's Fit to Print"

The New York Times

Late Edition

New York: Today, partly sunny, milder. High 59-64. Tonight, mostly cloudy. Low 48-54. Tomorrow, cloudy, windy, rain developing. High 57-62. Yesterday: High 58, low 41. Details, page D16.

VOL.CXXXVIII . . . No. 47,679 Copyright © 1988 The New York Times NEW YORK, FRIDAY, NOVEMBER 4, 1988 50 cents beyond 75 miles from New York City, except on Long Island. 35 CENTS

The New York Times/Jim Wilson

Gov. Michael S. Dukakis having his picture taken by a 10-year-old fan at a town meeting in Fairless Hills, Pa., during a tour of the Northeast in which he emphasized the drug problem. Page A19. Vice President Bush addressed supporters a rally in Columbus, Ohio. Less than a week after Mr. Dukakis acknowledged being a liberal, Mr. Bush said yesterday that "this election is not about labels." Page A18.

Registration Off Since 1984 Vote

There has been a pronounced decline in the percentage of eligible Americans who are registered to vote, a research group reports.

Nationally, the percentage of eligible Americans who are registered is estimated to be 78.9-percent, down 2.2 points from the 1984 level.

The group's study concluded that in many of the 30 states where final figures are available the decline was among

'Virus' in Military Computers Disrupts Systems Nationwide

By JOHN MARKOFF

In an intrusion that raises questions about the vulnerability of the nation's computers, a Department of Defense network has been disrupted since Wednesday by a rapidly spreading "virus" program apparently introduced by a computer science student.

The program reproduced itself through the computer network, making hundreds of copies in each machine it reached, effectively clogging systems linking thousands of military, corporate and university computers around the nation and preventing them from doing additional work. The virus is thought not to have destroyed any files.

By late yesterday afternoon computer experts were calling the virus the largest assault ever on the nation's computers.

'The Big Issue'

"The big issue is that a relatively benign software program can virtually bring our computing community to its knees and keep it there for some time," said Chuck Cole, deputy computer security manager at Lawrence Livermore Laboratory in Livermore, Calif., one of the sites affected by the intrusion. "The cost is going to be staggering."

Clifford Stoll, a computer security expert at Harvard University, added: "There is not one system manager who is not tearing his hair out. It's causing enormous headaches."

The affected computers carry a tremendous variety of business and research information among military officials, researchers and corporations.

While some sensitive military data are involved, the computers handling the nation's most sensitive secret information, like that on the control of nuclear weapons, are thought not to have been touched by the virus.

Parallel to Biological Virus

Computer viruses are so named because they parallel in the computer world the behavior of biological viruses. A virus is a program, or a set of instructions to a computer, that is either planted on a floppy disk meant to be used with the computer or introduced when the computer is communicating over telephone lines or data networks with other computers.

The programs can copy themselves into the computer's master software, or operating system, usually without calling any attention to themselves. From there, the program can be passed to additional computers.

Depending upon the intent of the software's creator, the program might cause a provocative but otherwise harmless message to appear on the computer's screen. Or it could systematically destroy data in the computer's memory. In this case, the virus program did nothing more than reproduce itself rapidly.

The program was apparently a result of an experiment, which

Continued on Page A21, Column 2

PENTAGON REPORTS IMPROPER CHARGES FOR CONSULTANTS

CONTRACTORS CRITICIZED

Inquiry Shows Routine Billing of Government by Industry on Fees, Some Dubious

By JOHN H. CUSHMAN Jr.

Special to The New York Times

WASHINGTON, Nov. 3 — A Pentagon investigation has found that the nation's largest military contractors routinely charge the Defense Department for hundreds of millions of dollars paid to consultants, often without justification.

The report of the investigation said that neither the military's current rules nor the contractors' own policies are adequate to assure that the Government does not improperly pay for privately arranged consulting work. Senior Defense Department officials said the Pentagon was proposing changes to correct the flaws.

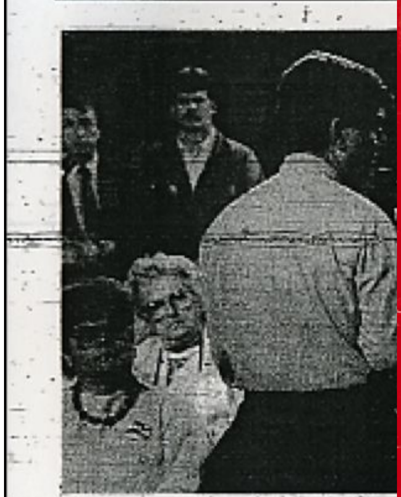
While it is not improper for military contractors to use consultants in performing work for the Pentagon, the work must directly benefit the military if it is to be paid for by the Defense Department. Often, Pentagon investigators discovered, this test is not met.

Broader Look at Consultants

The Justice Department's continuing criminal investigation has focused attention on consultants and their role in the designing and selling of weapons, and the Defense Department has been criticized for using consultants too freely. Now the Pentagon's own inves-

"All the News
That's Fit to Print"

VOL.CXXXVIII . . . No. 47,679



Gov. Michael S. Dukakis having his picture taken a 10-year-old fan at a town meeting in Fairless Hi Pa., during a tour of the Northeast in which he e phasized the drug problem. Page A19. Vice Pre

Registration Off Since 1984 Vote

There has been a pronounced decline in the percentage of eligible Americans who are registered to vote, a research group reports.

Nationally, the percentage of eligible Americans who are registered is estimated to be 78.3-percent, down 2.2 points from the 1984 level.

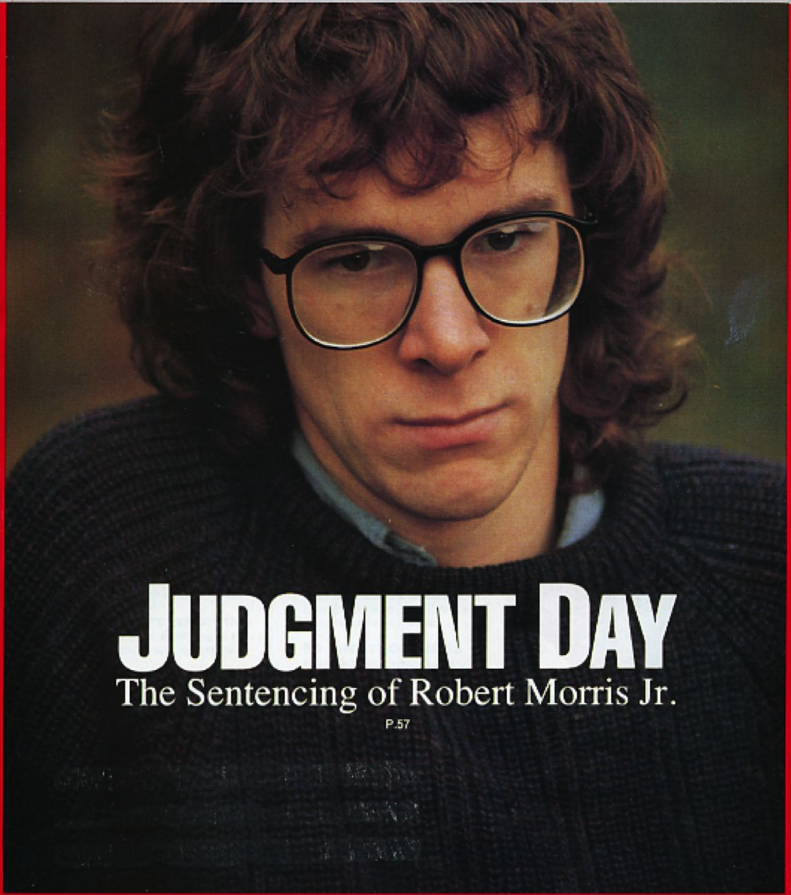
The group's study concluded that in many of the 30 states where final figures are available the decline was among

MAY 7, 1990

INFORMATIONWEEK®

THE NEWSMAGAZINE FOR INFORMATION MANAGEMENT

A CMP PUBLICATION $3.00



JUDGMENT DAY

The Sentencing of Robert Morris Jr.

P.57

Late Edition

New York: Today, partly sunny, milder. High 59-64. Tonight, mostly cloudy. Low 48-54. Tomorrow, cloudy, windy, rain developing. High 57-62. Yesterday: High 58, low 41. Details, page D16.

New York City, except on Long Island. 35 CENTS

PENTAGON REPORTS IMPROPER CHARGES FOR CONSULTANTS

CONTRACTORS CRITICIZED

Inquiry Shows Routine Billing of Government by Industry on Fees, Some Dubious

By JOHN H. CUSHMAN Jr.
Special to The New York Times

WASHINGTON, Nov. 3 — A Pentagon investigation has found that the nation's largest military contractors routinely charge the Defense Department for hundreds of millions of dollars paid to consultants, often without justification.

The report of the investigation said that neither the military's current rules nor the contractors' own policies are adequate to assure that the Government does not improperly pay for privately arranged consulting work. Senior Defense Department officials said the Pentagon was proposing changes to correct the flaws.

While it is not improper for military contractors to use consultants in performing work for the Pentagon, the work must directly benefit the military if it is to be paid for by the Defense Department. Often, Pentagon investigators discovered, this test is not met.

Broader Look at Consultants

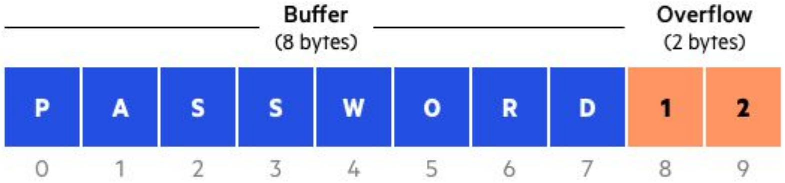
The Justice Department's continuing criminal investigation has focused attention on consultants and their role in the designing and selling of weapons, and the Defense Department has been criticized for using consultants too freely. Now the Pentagon's own inves-

How to take down the internet (in 1988)

1. Many programs were not “memory-safe” back then
   a. Programs would let you access memory on the computer that you shouldn’t have access to

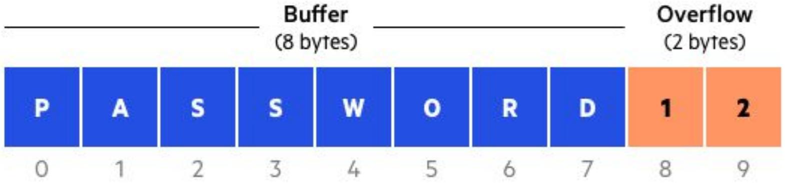
How to take down the internet (in 1988)

1. Many programs were not “memory-safe” back then
   a. Programs would let you access memory on the computer that you shouldn’t have access to
2. Find an array/buffer that lets you access memory you shouldn’t have access to



How to take down the internet (in 1988)

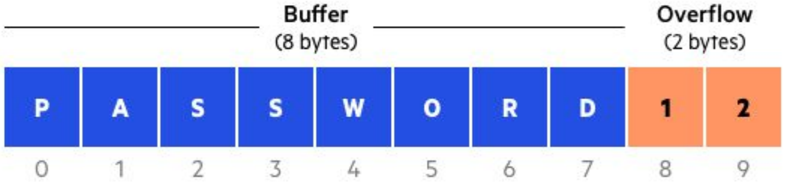
1. Many programs were not “memory-safe” back then
   a. Programs would let you access memory on the computer that you shouldn’t have access to
2. Find an array/buffer that lets you access memory you shouldn’t have access to



3. Inject some malicious code right after that array
   a. The computer will get tricked into running the code

How to take down the internet (in 1988)

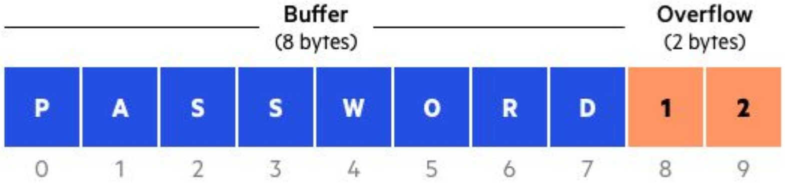
1. Many programs were not “memory-safe” back then
   a. Programs would let you access memory on the computer that you shouldn’t have access to
2. Find an array/buffer that lets you access memory you shouldn’t have access to



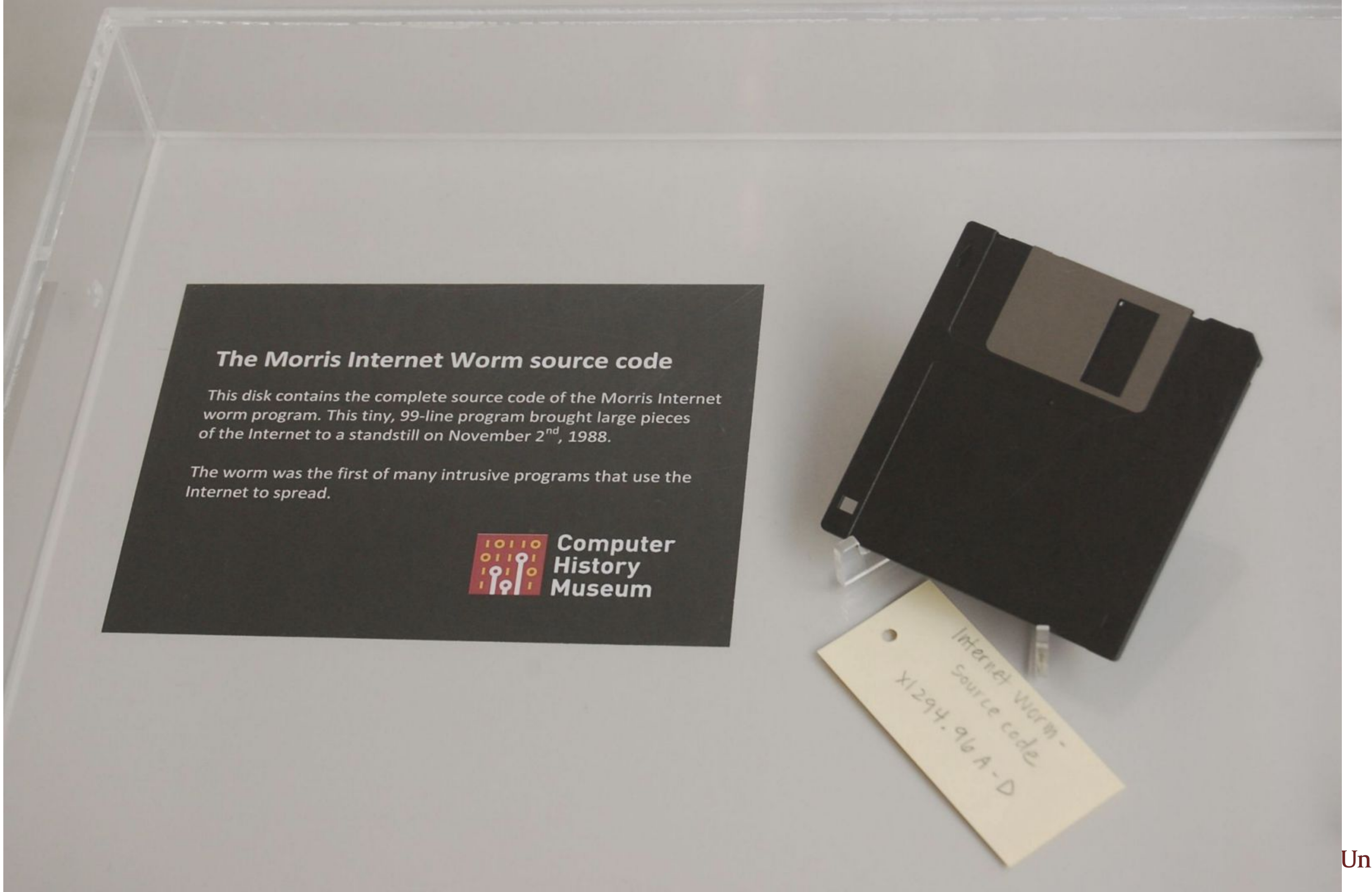
3. Inject some malicious code right after that array
   a. The computer will get tricked into running the code
4. Accidentally add a bug that eats up all of the memory on each host computer

How to take down the internet (in 1988)

1. Many programs were not “memory-safe” back then
   a. Programs would let you access memory on the computer that you shouldn’t have access to
2. Find an array/buffer that lets you access memory you shouldn’t have access to



3. Inject some malicious code right after that array
   a. The computer will get tricked into running the code
4. Accidentally add a bug that eats up all of the memory on each host computer
5. Crash the entire internet



The Morris Internet Worm source code

This disk contains the complete source code of the Morris Internet worm program. This tiny, 99-line program brought large pieces of the Internet to a standstill on November 2nd, 1988.

The worm was the first of many intrusive programs that use the Internet to spread.

Computer
History
Museum

"Responsible" Hacking

- The story of Robert Morris and his Internet Worm illustrates the core dilemma at the heart of security research
- Identifying and exposing security vulnerabilities is very important!
- Exposing security vulnerabilities in an irresponsible manner can result in devastating damages (monetary, physical, etc.)
- Responsible Disclosure: a vulnerability disclosure model in which a vulnerability or an issue is disclosed only after a period of time that allows for the vulnerability or issue to be patched or mended.

Memory on Stack vs Heap

```
Vector<string> varOnStack;
```

- Until today, all variables we've created get defined on the stack
- This is static memory allocation
- Variables on the stack are stored directly to the memory and access to this memory is very fast
- We don't have to worry about memory management

Memory on Stack vs Heap

```
Vector<string> varOnStack;
```

- Until today, all variables we've created get defined on the stack
- This is static memory allocation
- Variables on the stack are stored directly to the memory and access to this memory is very fast
- We don't have to worry about memory management

```
string* arr = new string[numValues];
```

Memory on Stack vs Heap

```
Vector<string> varOnStack;
```

- Until today, all variables we've created get defined on the stack
- This is static memory allocation
- Variables on the stack are stored directly to the memory and access to this memory is very fast
- We don't have to worry about memory management

```
string* arr = new string[numValues];
```

- We can now request memory from the heap
- This is dynamic memory allocation
- We have more control over variables on the heap
- But this means that we also have to handle the memory we're using carefully and properly clean it up when done

Cleaning Up

- When declaring local variables or parameters, C++ automatically handles memory allocation and deallocation for you
    - Memory allocation is the process by which the computer hands you a piece of computer memory in which you can store data
    - Memory deallocation is the process by which control of this memory (data storage location) is relinquished back to the computer

Cleaning Up

- When declaring local variables or parameters, C++ automatically handles memory allocation and deallocation for you
- When using new, you are responsible for deallocating the memory you allocate

Cleaning Up

- When declaring local variables or parameters, C++ automatically handles memory allocation and deallocation for you
- When using new, you are responsible for deallocating the memory you allocate
- If you don't, you get a memory leak

Cleaning Up

- When declaring local variables or parameters, C++ automatically handles memory allocation and deallocation for you
- When using new, you are responsible for deallocating the memory you allocate
- If you don't, you get a memory leak
  - Your program will never be able to use that memory again

Cleaning Up

- When declaring local variables or parameters, C++ automatically handles memory allocation and deallocation for you
- When using new, you are responsible for deallocating the memory you allocate
- If you don't, you get a memory leak
  - Your program will never be able to use that memory again
  - Too many leaks can cause a program to crash – it’s important to not leak memory!

Cleaning Up: delete

- You can deallocate (free) memory with the delete keyword
- To deallocate a single element:

```
delete var;
```

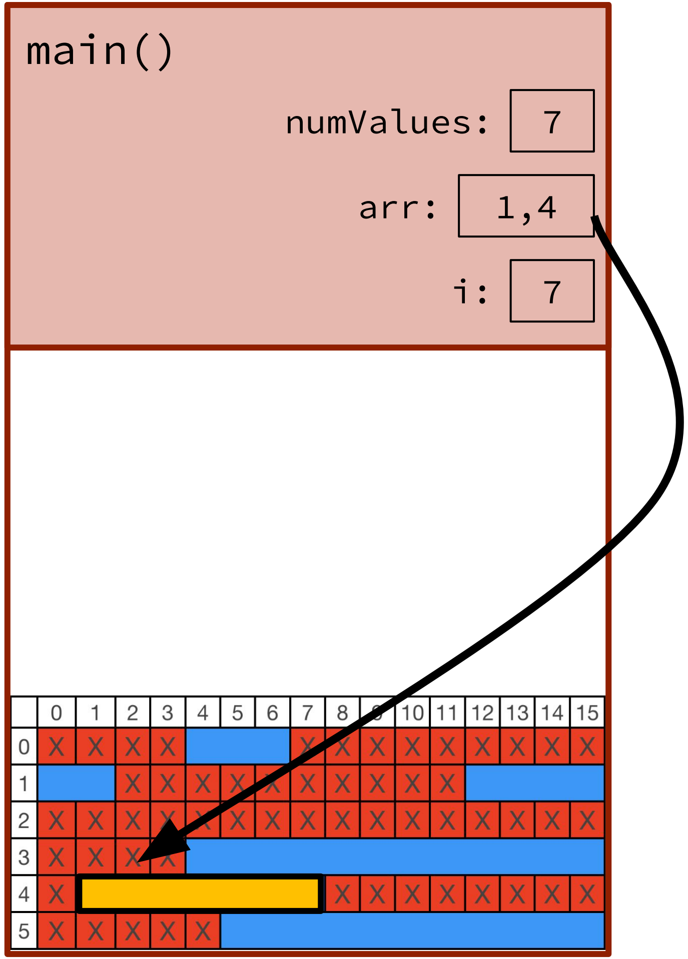
- To deallocate an array of elements:

```
delete[] arr;
```

Cleaning Up: `delete`

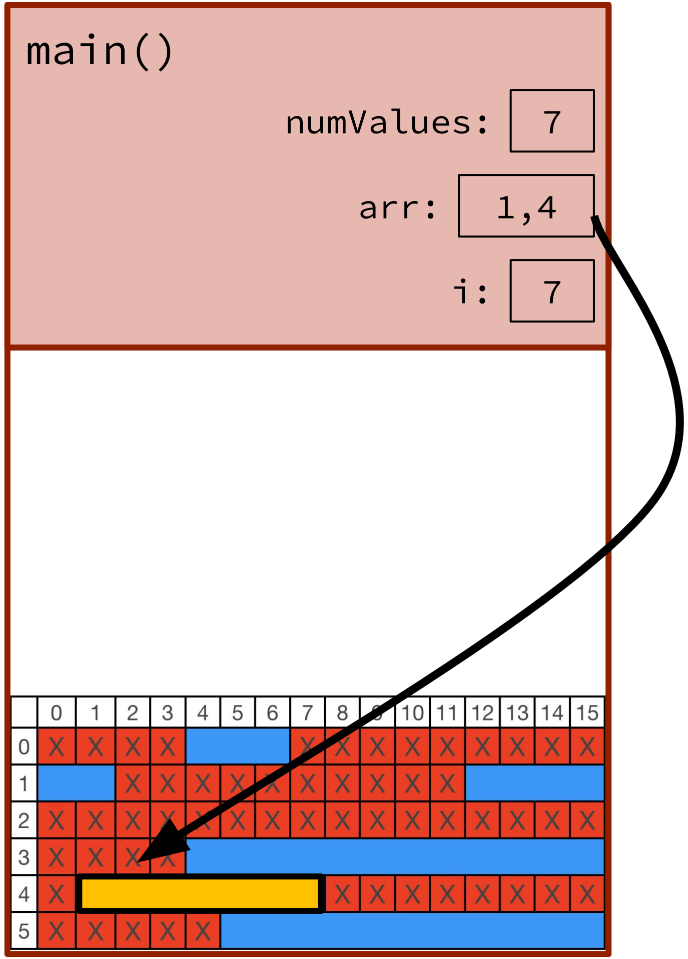
- This destroys the array pointed to by the given pointer, not the pointer itself
  - You can think of this operation as relinquishing control over the memory back to the computer
- Once you’ve deleted the memory pointed at by a pointer, you have a dangling pointer and shouldn’t read or write from it.

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  return 0;
}
```



Watch	me	dance,	dance	the	night	away!
0	1	2	3	4	5	6

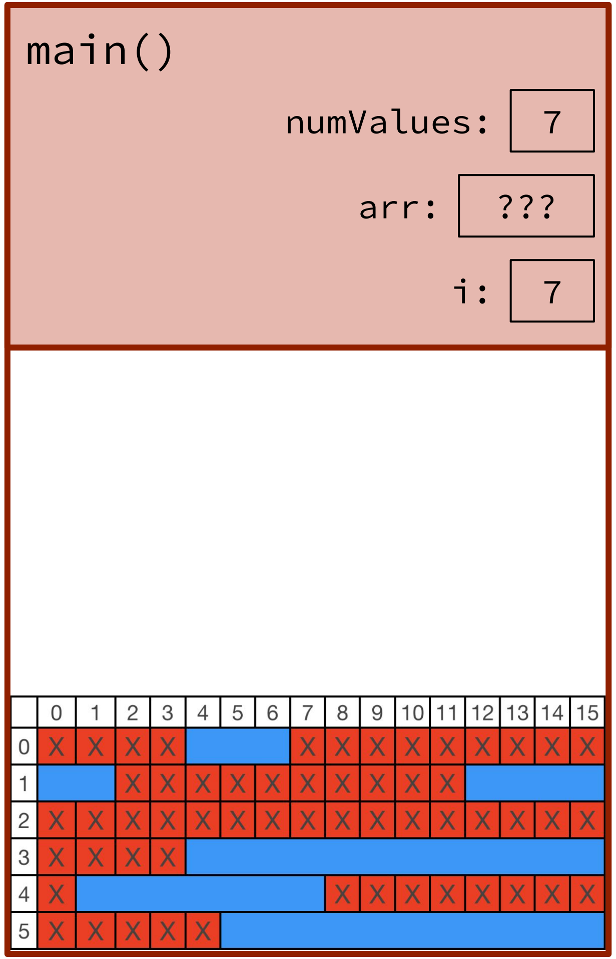
```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  delete[] arr;
  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

Watch	me	dance,	dance	the	night	away!
0	1	2	3	4	5	6

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  delete[] arr;
  return 0;
}
```



main()

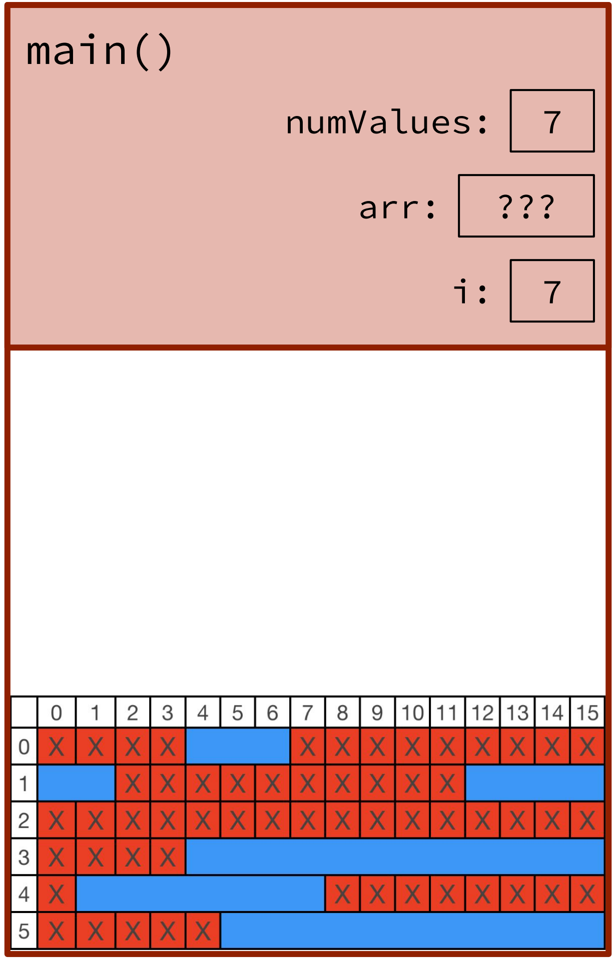
numValues: 7

arr: ???

i: 7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  delete[] arr;

  arr = new string[10];
  arr[4] = “weird”;
  delete[] arr;

  return 0;
}
```



main()

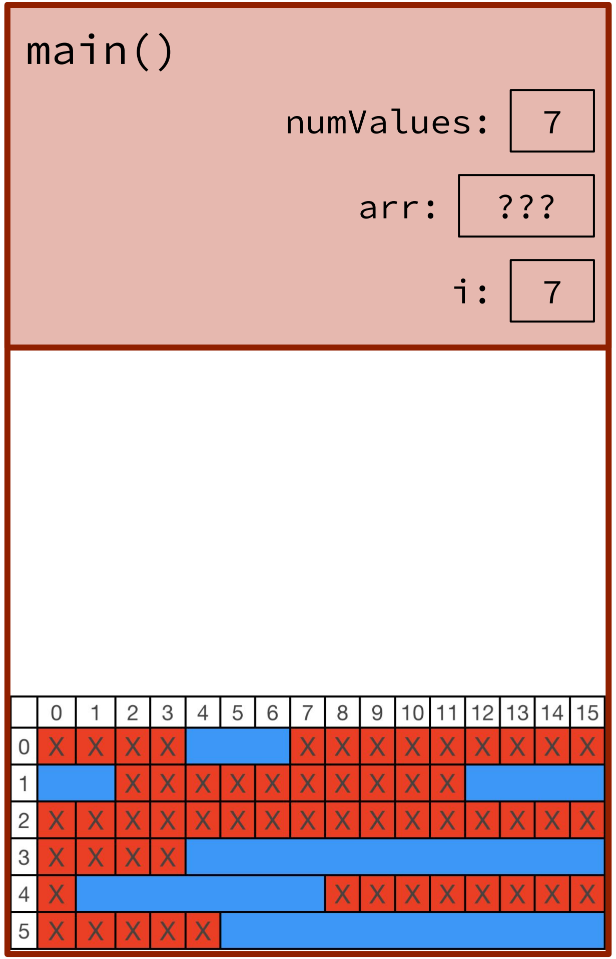
numValues: 7

arr: ???

i: 7

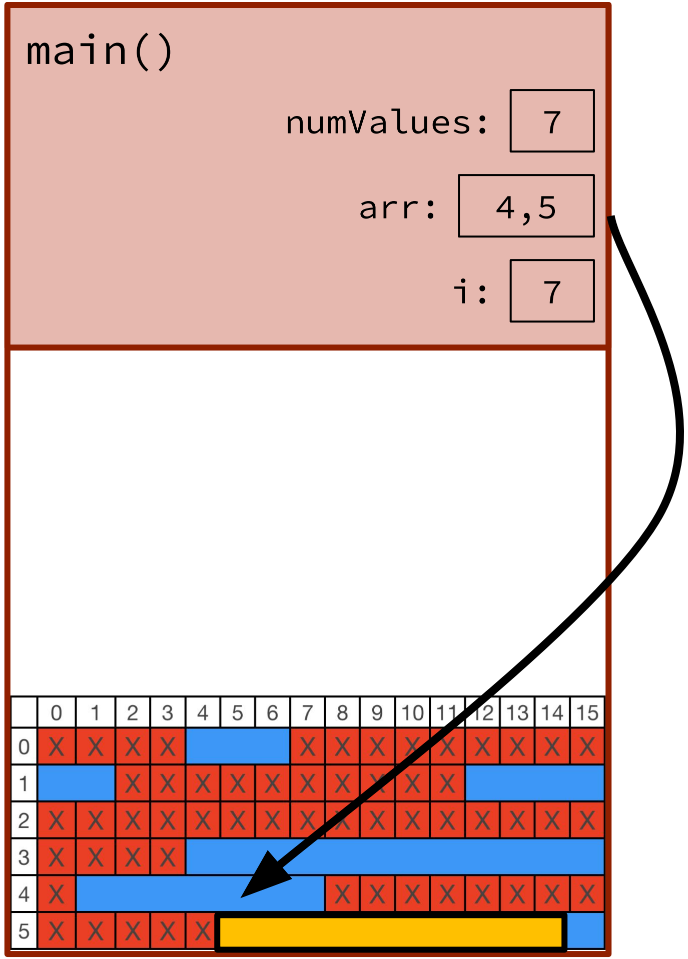
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  delete[] arr;

  arr = new string[10];
  arr[4] = “weird”;
  delete[] arr;

  return 0;
}
```



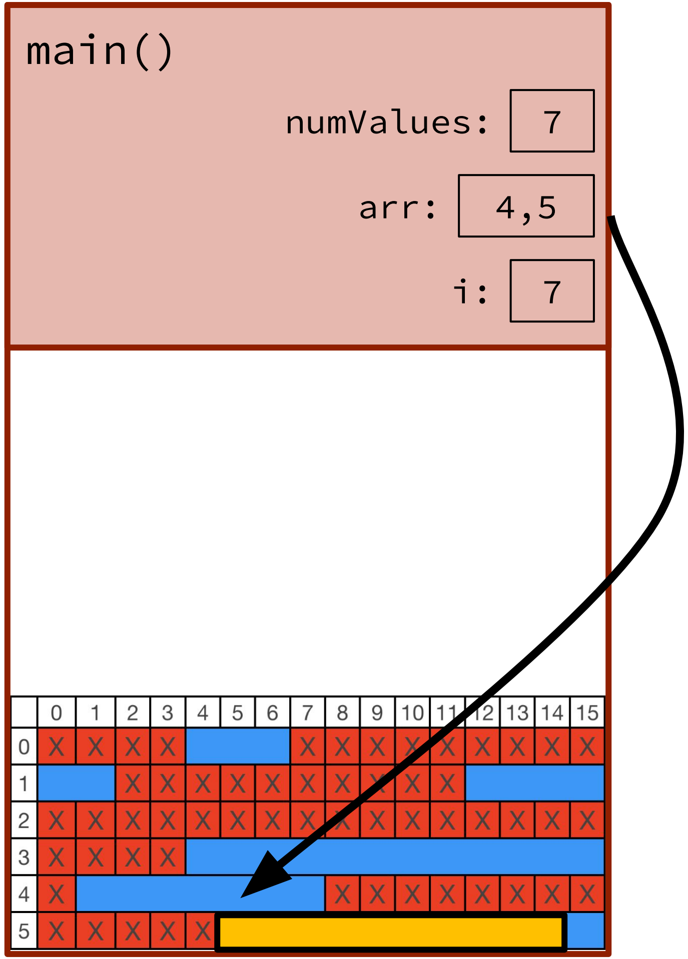
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  delete[] arr;

  arr = new string[10];
  arr[4] = “weird”;
  delete[] arr;

  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  delete[] arr;

  arr = new string[10];
  arr[4] = “weird”;
  delete[] arr;

  return 0;
}
```



main()

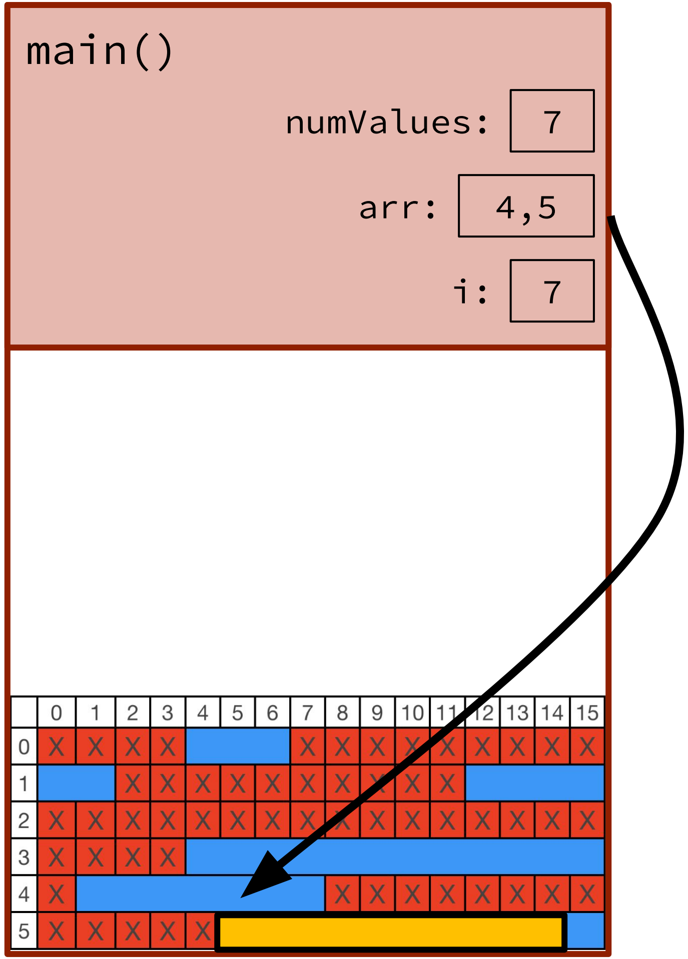
numValues: 7

arr: 4,5

i: 7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  delete[] arr;

  arr = new string[10];
  arr[4] = “weird”;
  delete[] arr;

  return 0;
}
```



main()

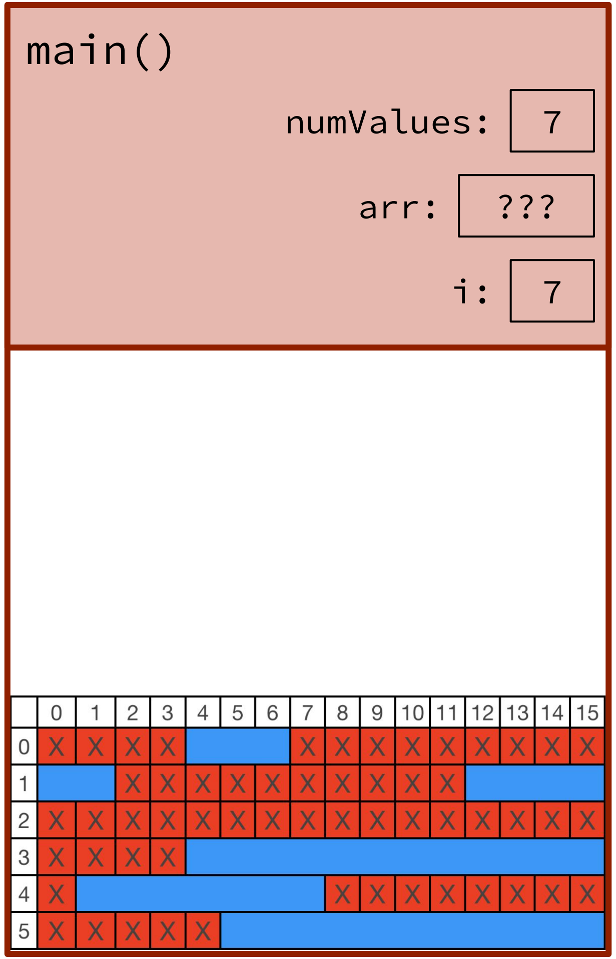
numValues: 7

arr: 4,5

i: 7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟦

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  delete[] arr;

  arr = new string[10];
  arr[4] = “weird”;
  delete[] arr;

  return 0;
}
```



main()

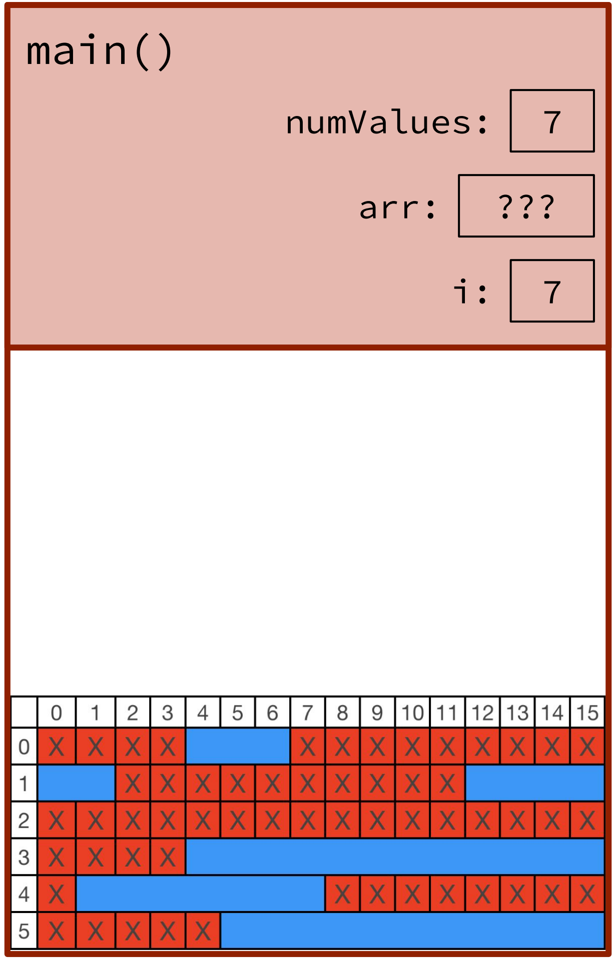
numValues: 7

arr: ???

i: 7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X	🟦	🟦	🟦	X	X	X	X	X	X	X	X	X
1	🟦	🟦	X	X	X	X	X	X	X	X	X	X	🟦	🟦	🟦	🟦
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	X	X	X	X	X	X	X	X
5	X	X	X	X	X	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

```
int main () {
  int numValues = getInteger("How many words?");
  string* arr = new string[numValues];
  for (int i = 0; i < numValues; i++) {
    arr[i] = getLine("Enter a string: ");
  }
  for (int i = 0; i < numValues; i++) {
    cout << i << ": " << arr[i] << endl;
  }
  delete[] arr;

  arr = new string[10];
  arr[4] = “weird”;
  delete[] arr;

  cout << arr[1] << endl; // DO NOT DO THIS
  arr = new int[4]; // ERROR

  return 0;
}
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	X	X				X	X	X	X	X	X	X	X	X
1			X	X	X	X	X	X	X	X	X	X				
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3	X	X	X	X												
4	X								X	X	X	X	X	X	X	X
5	X	X	X	X	X											

Recap

- We've learned about classes, which have an **interface** and **implementation**.
- When implementing classes at the lowest level of abstraction, we need to use **dynamic memory** as a fundamental building block for specifying how much memory something needs.
    - We use the keyword **`new`** to allocate dynamic memory.
    - We keep track of that memory with a **pointer**. (more on pointers Thursday!)
    - We must clean up the memory when we're done with **`delete`**.
- We've learned how to allocate dynamic memory using **arrays**, which give us a contiguous block of memory that all stores one particular type (int, string, double, etc.).
- Without knowing it, we have been using dynamic memory all along, through the use of the standard and Stanford library classes. The string, Vector, Map, Set, Stack, Queue, etc., all use dynamic memory to give you the data structures we have used for all our programs.

Next Class - Implementing a Dynamic ADT

We’re going to build a vector!