CIS 190: C/C++ Programming

Lecture 12

Student Choice

Outline

- Hash Maps
  - Collisions
  - Using
    - Open Addressing Collisions
    - Chaining Collisions
  - In C++
- C++ STL
  - Containers
- C++ GUI Resources

Hash Maps (AKA Hash Tables)

- data structure that maps *keys* to *values*
- a *hash function* takes a given key and outputs an *index* into an array, where the value will be stored
- providing the same key will produce the same index, where the value is stored

Hash Map Example

- **key**: name
- **value**: phone number

0	867-5309
1	555-2368
2	
3	

- the **`Hash()`** function gives an integer; then use modulus to get a valid index for the table

```
Hash("Jenny") = 68;     68 % 4 = 0;
Hash("Egon") =  41;     41 % 4 = 1;
```

Choosing a Hash Function

- a good hash function is
  - easy to compute
  - has a uniform distribution of keys

- hash function uniformity is dependent on the size of the hash map
  - powers of two
  - prime numbers

Why Use Hash Maps?

- speed
- in best-case scenario, the lookup time is O(1); the requested value is found immediately
- in worst-case scenario (collisions), lookup time can be O(n)
  - this scenario is incredibly rare

Outline

- Hash Maps
  - Collisions
  - Using
    - Open Addressing Collisions
    - Chaining Collisions
  - In C++
- C++ STL
  - Containers
- C++ GUI Resources

Collisions

- occur when two keys map to the same index

- many ways of handling, dependent on situation
  - chaining
  - open addressing
  - etc.

Hash Map Example – Setup

- **key**: class number
- **value**: class name

- examples:
  - key: 190 value: C++ Prog.
  - key: 191 value: Python

- have less than a dozen entries, so our hash map size will be 11 (prime number)

Hash Map Example – Function

- the Hash() function we're going to use is very naïve and not very good
- the function is very simple: the digits of the key are multiplied together, excepting zeroes
- so Hash(123) = 1 * 2 * 3 = 6

Outline

- Hash Maps
  - Collisions
  - Using
    - Open Addressing Collisions
    - Chaining Collisions
  - In C++
- C++ STL
  - Containers
- C++ GUI Resources

Hash Map Example – Using

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

- let's add "190 - C++ Prog."

Hash Map Example – Using

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

- let's add "190 - C++ Prog."
  - Hash(190) = 1 * 9 = **9**

Hash Map Example – Using

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	C++ Prog.
10	

- let's add "190 - C++ Prog."
  - Hash(190) = 1 * 9 = **9**

Hash Map Example – Using

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	C++ Prog.
10	

- let's add "190 - C++ Prog."
  - Hash(190) = 1 * 9 = **9**
- and also "240 - Comp Arch."
  - Hash(240) = 2 * 4 = **8**

Hash Map Example – Using

0	
1	
2	
3	
4	
5	
6	
7	
8	Comp Arch.
9	C++ Prog.
10	

- let's add "190 - C++ Prog."
  - Hash(190) = 1 * 9 = **9**
- and also "240 - Comp Arch."
  - Hash(240) = 2 * 4 = **8**

Hash Map Example – Using

0	
1	
2	
3	
4	
5	
6	
7	
8	Comp Arch.
9	C++ Prog.
10	

- let's add "190 - C++ Prog."
  - Hash(190) = 1 * 9 = **9**
- and also "240 - Comp Arch."
  - Hash(240) = 2 * 4 = **8**
- and then "262 - Automata"
  - Hash(262) = 2 * 6 * 2 = **24**

Hash Map Example – Using

0	
1	
2	
3	
4	
5	
6	
7	
8	Comp Arch.
9	C++ Prog.
10	

- let's add "190 - C++ Prog."
  - Hash(190) = 1 * 9 = **9**
- and also "240 - Comp Arch."
  - Hash(240) = 2 * 4 = **8**
- and then "262 - Automata"
  - Hash(262) = 2 * 6 * 2 = **24**
  - 24 is too large, so we'll use mod: 24 % 11 = **2**

Hash Map Example – Using

0	
1	
2	Automata
3	
4	
5	
6	
7	
8	Comp Arch.
9	C++ Prog.
10	

- let's add "190 - C++ Prog."
  - Hash(190) = 1 * 9 = **9**
- and also "240 - Comp Arch."
  - Hash(240) = 2 * 4 = **8**
- and then "262 - Automata"
  - Hash(262) = 2 * 6 * 2 = **24**
  - 24 is too large, so we'll use mod: 24 % 11 = **2**

Hash Map Example – Using

0	
1	
2	Automata
3	
4	
5	
6	
7	
8	Comp Arch.
9	C++ Prog.
10	

- so far so good!
- let's add "191 - Python"
  - Hash(191) = 1 * 9 * 1 = **9**

Hash Map Example – Using

0	
1	
2	Automata
3	
4	
5	
6	
7	
8	Comp Arch.
9	C++ Prog.
10	

- so far so good!
- let's add "191 - Python"
  - Hash(191) = 1 * 9 * 1 = **9**
- "Python" is colliding with "C++ Prog."

Hash Map Example – Decisions

0		
1		
2	262	Automata
3		
4		
5		
6		
7		
8	240	Comp Arch.
9	190	C++ Prog.
10		

- require that we have been storing the key with the value
- this is a decision you need to make when you first set up your hash map

Hash Map Example – Decisions

0		
1		
2	262	Automata
3		
4		
5		
6		
7		
8	240	Comp Arch.
9	190	C++ Prog.
10		

- now we can use open addressing or chaining

Outline

- Hash Maps
  - Collisions
  - Using
    - Open Addressing Collisions
    - Chaining Collisions
  - In C++
- C++ STL
  - Containers
- C++ GUI Resources

Collisions – Open Addressing

- key is stored with the value in the given index
- when a collision occurs, the hash table is probed until an empty index has been found
  - different probe sequences can be used
- using that same probe sequence, you can then find the given value when you use the same key

Hash Map Example – Open Addressing

0		
1		
2	262	Automata
3		
4		
5		
6		
7		
8	240	Comp Arch.
9	190	C++ Prog.
10		

- for open addressing, we need to choose a probe procedure

Hash Map Example – Open Addressing

0		
1		
2	262	Automata
3		
4		
5		
6		
7		
8	240	Comp Arch.
9	190	C++ Prog.
10		

- for open addressing, we need to choose a probe procedure
- for simplicity's sake, we'll use linear probing
  - interval of probes is fixed
  - we'll use an interval of 1

Hash Map Example – Open Addressing

0		
1		
2	262	Automata
3		
4		
5		
6		
7		
8	240	Comp Arch.
9	190	C++ Prog.
10	191	Python

- for open addressing, we need to choose a probe procedure
- for simplicity's sake, we'll use linear probing
  - interval of probes is fixed
  - we'll use an interval of 1

Hash Map Example – Open Addressing

0		
1		
2	262	Automata
3		
4		
5		
6		
7		
8	240	Comp Arch.
9	190	C++ Prog.
10	191	Python

- "120 – Prog. I"
  - Hash(120) = 1 * 2 = **2**

Hash Map Example – Open Addressing

0		
1		
2	262	Automata
3	120	Prog. I
4		
5		
6		
7		
8	240	Comp Arch.
9	190	C++ Prog.
10	191	Python

- "120 – Prog. I"
  - Hash(120) = 1 * 2 = **2**

Hash Map Example – Open Addressing

0		
1		
2	262	Automata
3	120	Prog. I
4		
5		
6		
7		
8	240	Comp Arch.
9	190	C++ Prog.
10	191	Python

- "120 – Prog. I"
  - Hash(120) = 1 * 2 = **2**
- "121 – Prog. II"
  - Hash(121) = 1 * 2 * 1 = **2**

Hash Map Example – Open Addressing

0		
1		
2	262	Automata
3	120	Prog. I
4		
5		
6		
7		
8	240	Comp Arch.
9	190	C++ Prog.
10	191	Python

- "120 – Prog. I"
  - Hash(120) = 1 * 2 = **2**
- "121 – Prog. II"
  - Hash(121) = 1 * 2 * 1 = **2**

Hash Map Example – Open Addressing

0		
1		
2	262	Automata
3	120	Prog. I
4	121	Prog. II
5		
6		
7		
8	240	Comp Arch.
9	190	C++ Prog.
10	191	Python

- "120 – Prog. I"
  - Hash(120) = 1 * 2 = **2**
- "121 – Prog. II"
  - Hash(121) = 1 * 2 * 1 = **2**

Outline

- Hash Maps
  - Collisions
  - Using
    - Open Addressing Collisions
    - Chaining Collisions
  - In C++
- C++ STL
  - Containers
- C++ GUI Resources

Collisions – Chaining

- key is stored with the value in the index's list

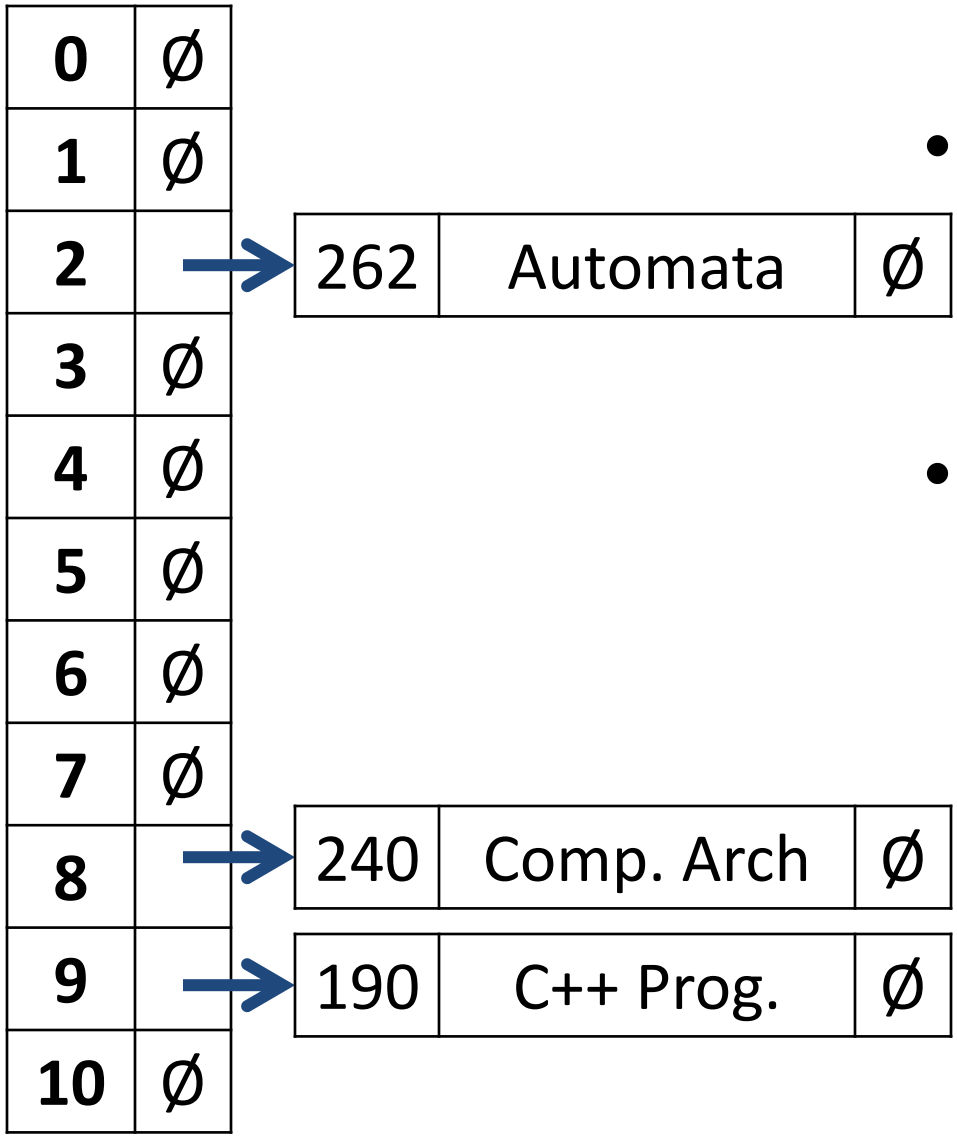
- each index has a list of entries
- when a collision occurs, the new value is added to the list
  - sorted
  - at end
  - at beginning

Hash Map Example – Chaining

0		
1		
2	262	Automata
3		
4		
5		
6		
7		
8	240	Comp Arch.
9	190	C++ Prog.
10		

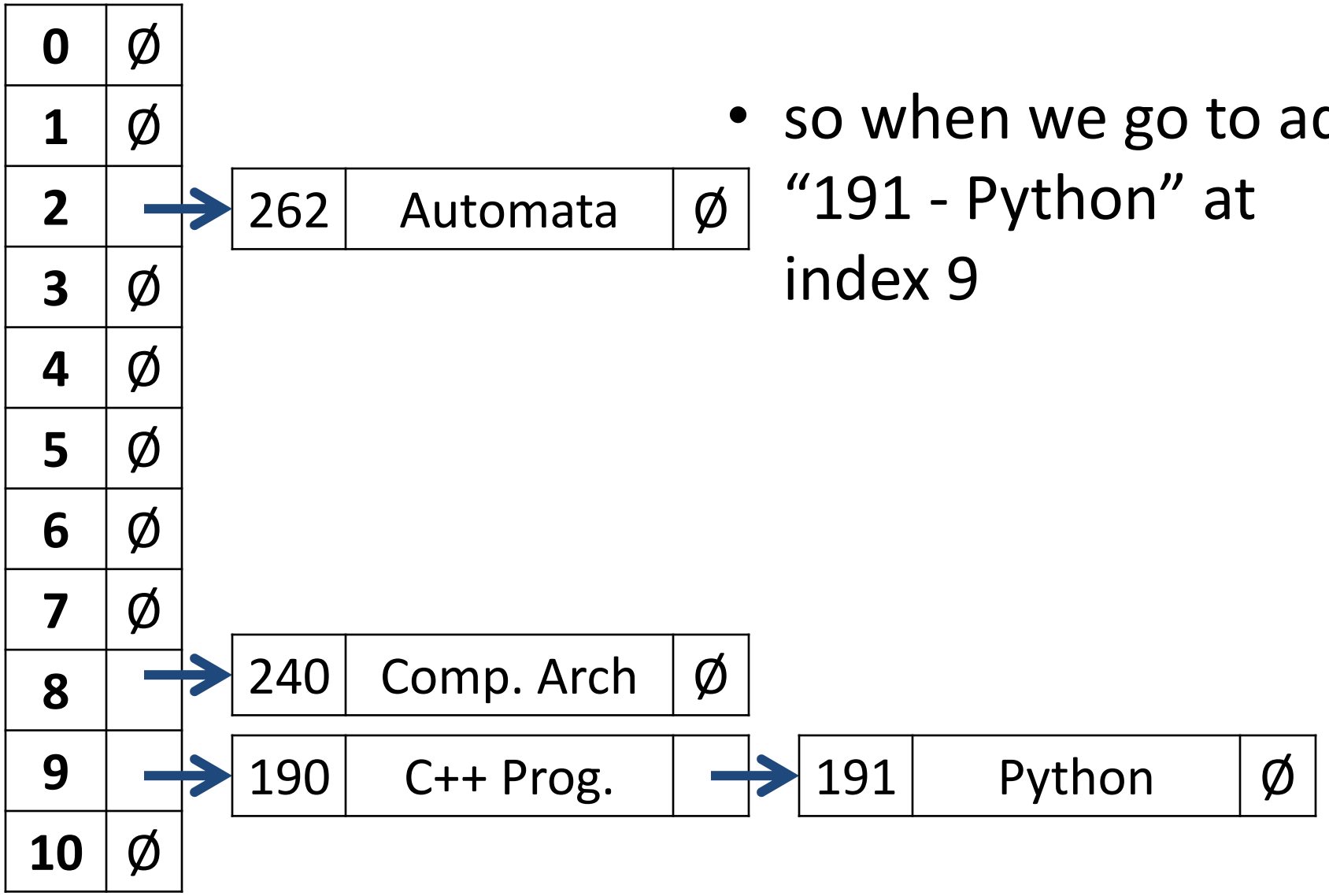
- chaining is even more different than open addressing
- in addition to storing keys with the values, the indexes instead contain **pointers** to a list of key/value pairs

Hash Map Example – Chaining



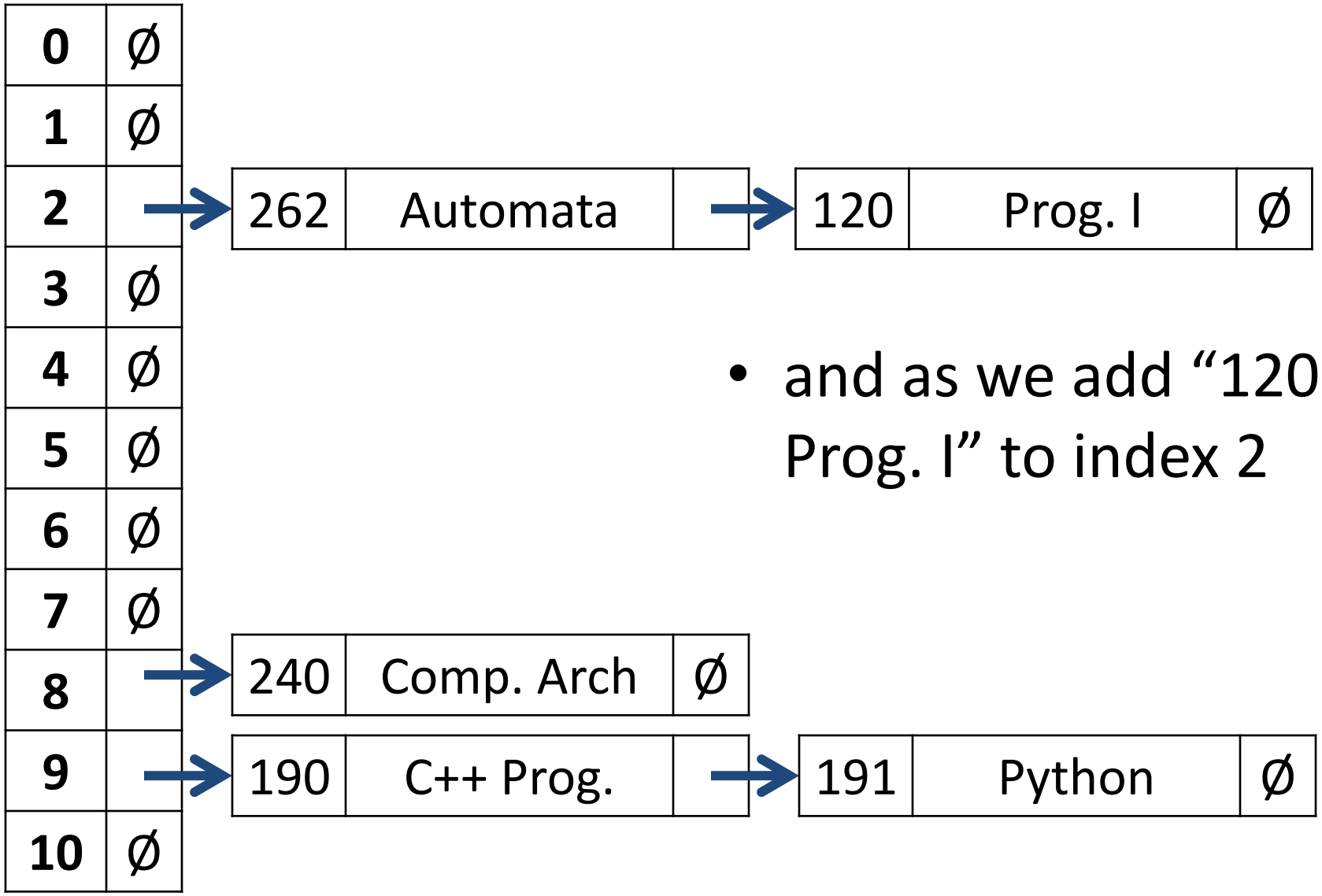
- chaining is even more different than open addressing
- in addition to storing keys with the values, the indexes instead contain **pointers** to a list of key/value pairs

Hash Map Example – Chaining



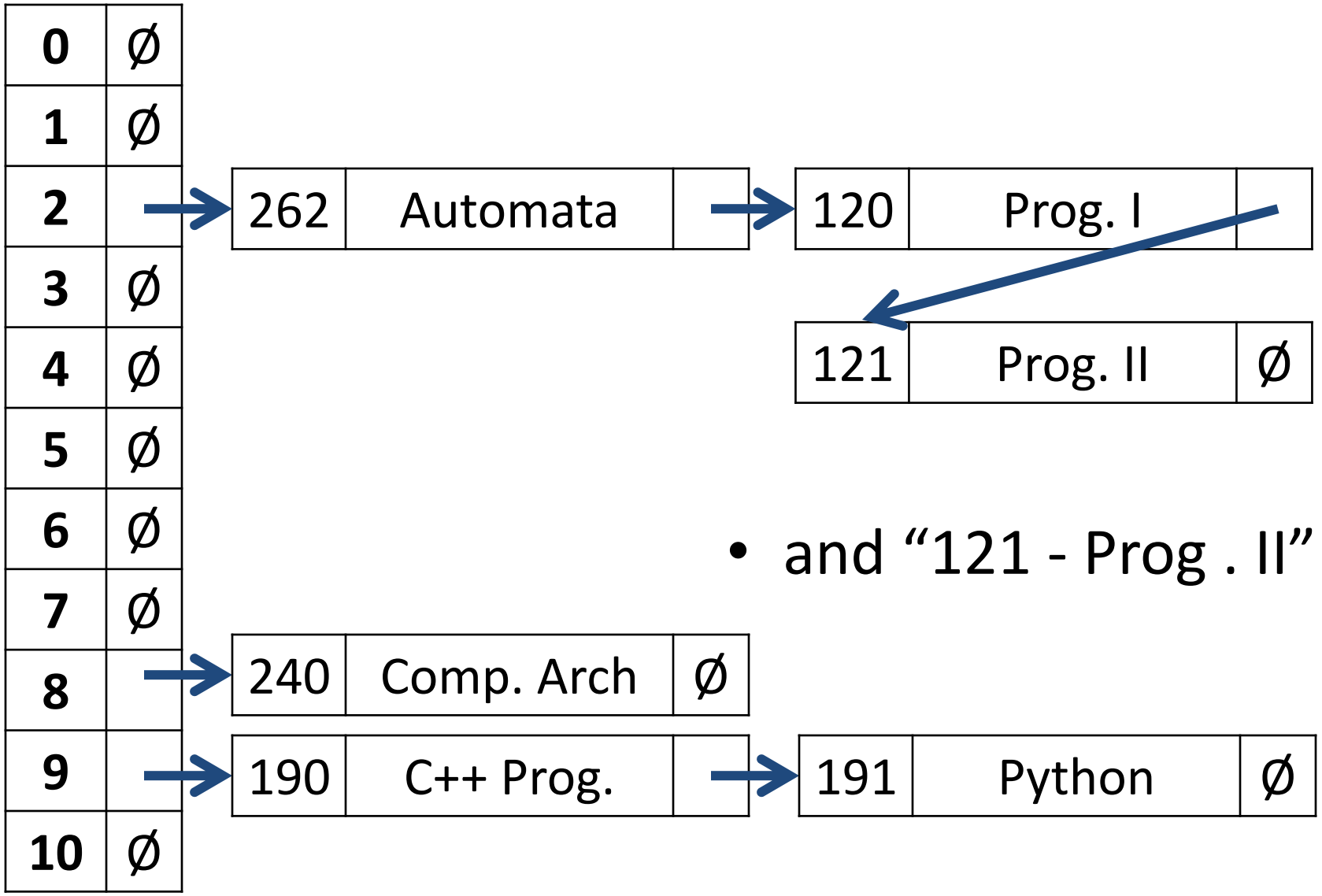
- so when we go to add "191 - Python" at index 9

Hash Map Example – Chaining



- and as we add “120 - Prog. I” to index 2

Hash Map Example – Chaining



- and “121 - Prog . II”

Hash Map Variations

- array of elements of needed size
  - alone
  - with elements allowing pointers to next in that index
- array of pointers
  - to (list of) elements
  - better in terms of space needed, if entries are larger than the size of a pointer

Drawbacks of Hash Maps

- collisions
- pseudo-random order
  - can't find next closest entry
- hash function may take a long time
- not very good for small numbers of things
- resizing can be a very slow operation
  - necessary to rehash all stored entries
    - means you need to store key with value as well

Outline

- Hash Maps
  - Collisions
  - Using
    - Open Addressing Collisions
    - Chaining Collisions
  - In C++
- C++ STL
  - Containers
- C++ GUI Resources

Hash Maps in C++

- there is technically a **`hash_map`** library
  - but it never made it into the STL
  - probably shouldn't rely on it (or use it)

- there are two standard STL containers you can use to help implement a hash map:
  - **`map`**
  - **`unordered_map`**

Map Containers

- **`map`** and **`unordered_map`** take a variable type for both key and value
  - **`map <string, int> mapInstance;`**

- maps are ordered by key
- unordered_maps are not
  - take a Hash function as an argument
  - unordered_maps require C++11

Outline

- Hash Maps
  - Collisions
  - Using
    - Open Addressing Collisions
    - Chaining Collisions
  - In C++
- C++ STL
  - Containers
- C++ GUI Resources

C++ STL

- Standard Template Library

- set of ready-made common classes
  - Iterators
  - Algorithms
  - Functors
  - Containers

C++ STL – Iterators

- iterators are used to traverse containers
- five types:
  - input, output, forward, bidirectional, random access
- iterators allow the STL to be flexible
  - can write a function using iterators that will work for both lists and vectors
- not always good for associative containers
  - have their own member functions for most things

C++ STL – Algorithms

- utility functions
  - searches
  - sorts
  - merges
  - partitioning
  - etc.

C++ STL – Functors

- also called function objects
- classes that overload the function call operator
  - `operator()`
- allows you to declare an object that can be work as and be treated as a function
  - can be passed into many STL functions

Outlines

- Hash Maps
  - Collisions
  - Using
    - Open Addressing Collisions
    - Chaining Collisions
  - In C++
- C++ STL
  - Containers
- C++ GUI Resources

C++ STL – Containers

- four different types of containers
  - sequence containers

  - **arrays** (**require C++11**)
  - vectors
  - deque FIFO & FILO capable
  - **forward_list** single-linked list (**C++11**)
  - list doubly-linked list

C++ STL – Containers

- four different types of containers
  - container adaptors
    - provide an interface that relies on a container class
  - stack FILO only
  - queue FIFO only
  - priority_queue modified queue
    - first element is always the largest

C++ STL – Containers

- four different types of containers
  - associative containers

  - map store keys and values together
  - set only store keys (value is the key)
  - multi- allow non-unique keys

  - use a compare function to sort elements by key

C++ STL – Containers

- four different types of containers
  - unordered associative containers (**require C++11**)
    - unordered_(multi)map/set
  - **`unordered_map`** will work as a hash map!
    - can take a key, a value, and a hash function
- hash maps are inherently unordered, which is why a regular **`map`** container won't work

Outline

- Hash Maps
  - Collisions
  - Using
    - Open Addressing Collisions
    - Chaining Collisions
  - In C++
- C++ STL
  - Containers
- C++ GUI Resources

C++ GUI – Intro

- GUI – Graphical User Interface
- before now, we've been using a CLI, or a Command Line Interface
- many GUI libraries/toolkits (for C++ and other languages) are for a specific operating system
  - avoid these, they tie you to a single OS

C++ GUI – Qt

- one of the most popular software dev packages for creating applications with GUIs
  - works across multiple platforms
- Qt uses standard C++
  - also has implementations for SQL databases, XML, interacting with multimedia, etc.
  - IDE is called Qt Creator; cross-platform
- qt-project.org

Project

- Alphas due Monday at midnight!
  - ***DO NOT TURN YOUR ALPHAS IN LATE***
  - will grade the most recent submission

- your alphas and final project **do not** have to run (or compile) on eniac
  - however, you still must submit all your files, including a Makefile