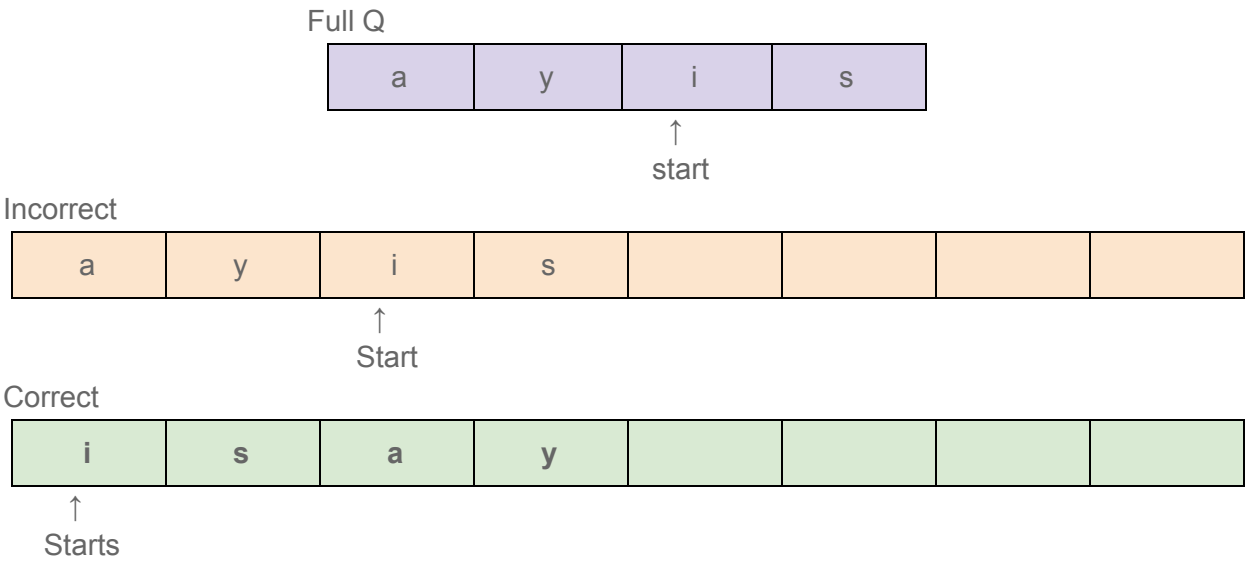
CS 225 Spring 2019 :: TA Lecture Notes
2/11 Iterators

By Wenjie

- **Queue Implementation**
  - We can implement queue using an array list.
  - Adding:
    - If we add an element to the array in front, we need to copy every element one spot up to create space for the new element in front.
    - Previously we said handle “add in front” efficiently by using smart indexing.
    - Lastly we need to increment capacity of the queue
  - When we are out of space:
    - We can double the array
    - But be careful how we copy the data over - need to copy data such that the start of the queue is at the index = 0.



Three data storage strategies

- When we talked about the functions we said that we can pass/return by value, reference, and pointer. The same concept applies to storing data. We can store data by value, reference, and pointer.

CS 225 Spring 2019 :: TA Lecture Notes 2/11 Iterators

By Wenjie

```
5  T & data; // store by reference
6
7  T * data; // store by pointer
8
9  T data; // store by value
```

There are tradeoffs among the three:

- Data life cycle management:
    - *By reference* - a reference is an alias to a variable, therefore storing a reference means that we do not own that data. I is somewhere on client's stack and when the client code finishes, the data is going to be deleted.
    - *By pointer* - a pointer has it's on memory, but similarly to the reference it points to a memory space that we do not control. The client code owns the memory and when it finishes the memory is going to be freed.
    - *By value* - in this case, we are going to get a copy of the object and we are completely in charge of the data. The only time it can get deleted is when our code is done or deleting the data.
- Storing NULL as the data:
    - *By reference* - a reference can never be a null, therefore we cannot have NULL stored.
    - *By pointer* - a pointer can hold NULL value and we can store it as data.
    - *By value* - when we create an object, either a custom constructors or a default automatic constructor will be called. Therefore, the data will never be NULL.
- Effects on stored data when the data is manipulated from user code (safety):
    - *By reference* - our data is an alias to the data in the client code, thereby when the client code changes the data, the data we store gets changed as well.
    - *By pointer* - our data points to the memory block that belongs to the client code. Therefore, as in the case of the reference, if the client code changes the data, the data we store gets changed as well.
    - *By value* - the data we store belongs to us and the changes made by the client code do not affect what we have stored.
- The relative speed of storage methods:
    - *By reference* - we are only storing an address. We don't need more memory and we don't need to use a copy constructor. Therefore, this is a fast method.

CS 225 Spring 2019 :: TA Lecture Notes 2/11 Iterators

By Wenjie

- *By pointer* - as for the reference, we are storing an address and this is a fast method.
- *By value* - in this case we need more memory to store the whole object and we need to use a copy constructor. Therefore, storing the data by value is slow.

Iterators

- Definition: Iterators are used to read the data from different data structures in a uniform way (a common interface). In other words, iterators maintain a list of data for client code. They are used to keep track of where we are, what is next, and what is the data in the data structure.
- Implementation:
  - In the implementing class
    - `::begin();` return an iterator that is at the beginning of the data
    - `::end();` return a iterator one past the last data
  - In the iterator class
    - base class: `std::iterator`
    - it requires us to implement basic functionality:
      - `operator++;` move to the next
      - `operator*;` dereferencing operator: returns data
      - `operator!=;` not equal operator: to check the end
- Using an iterator

stlList.cpp

```
1  #include <list>
2  #include <string>
3  #include <iostream>
4  struct Animal { // struct is a class with all members public
5    std::string name, food;
6    bool big;
7    Animal(std::string name = "blob", std::string food = "you", bool big
8  = true)
9      : name(name), food(food), big(big) {};
10     // constructor with default values
11     // this implicitly defines a default constructor
12   int main() {
```

CS 225 Spring 2019 :: TA Lecture Notes
2/11 Iterators

By Wenjie

```
13      Animal g("giraffe", "leaves", true), p("penguin", "fish", false),
14  b("bear");  //same as b("bear", "you", true);
15
16      std::vector zoo;
17      zoo.push_back(g);
18      zoo.push_back(p);  // std::vector's insertAtEnd
19      zoo.push_back(b);
20
21      for (std::vector<Animal>::iterator it = zoo.begin(); it !=
22  zoo.end(); it++) {
23        std::cout << (*it).name << " " << (*it).food << std::endl;
24      }
25
26  /*
27      // intead, we can use "for each loop"
28      for (const Animal & animal : zoo) {
29        std::cout << animal.name << " " << animal.food << std::endl;
30      }
31  */
32      return 0;
33  }
```

- For each loop
  - Always const: we are not be able to modify the variable
  - We do not care what the data structure is when using an iterator this way
    - If instead of vector, we use multimap, the for loop will be the same

```
1  for ( const TYPE & variable : collection ) {
2    // …
3  }
```