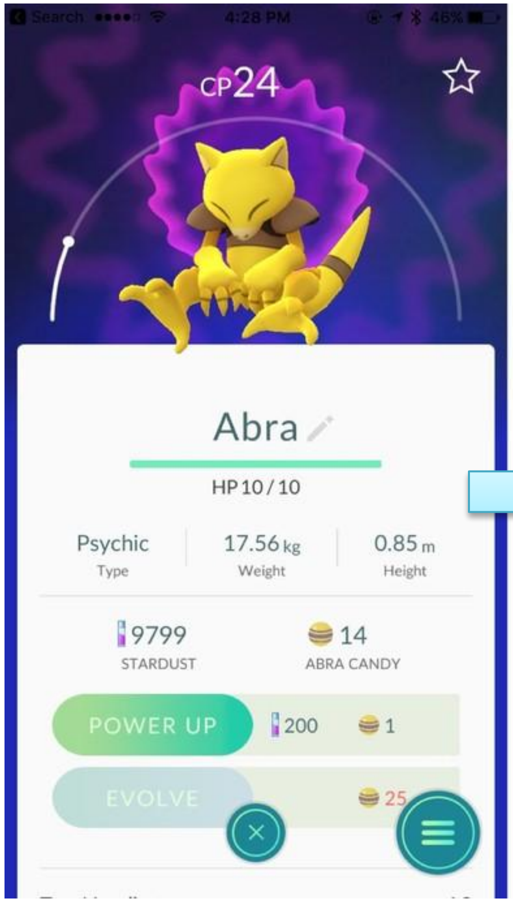
Heap and Queueing Simulation

Two BIG Parts

- Implementing Heap
  - Basic requirement
- Perform a queueing simulation experiment with Priority Queue
  - Difficult
  - Uses Heap as a Priority Queue
  - Treat it as a mini project

Time to Level UP!



CP24

Abra

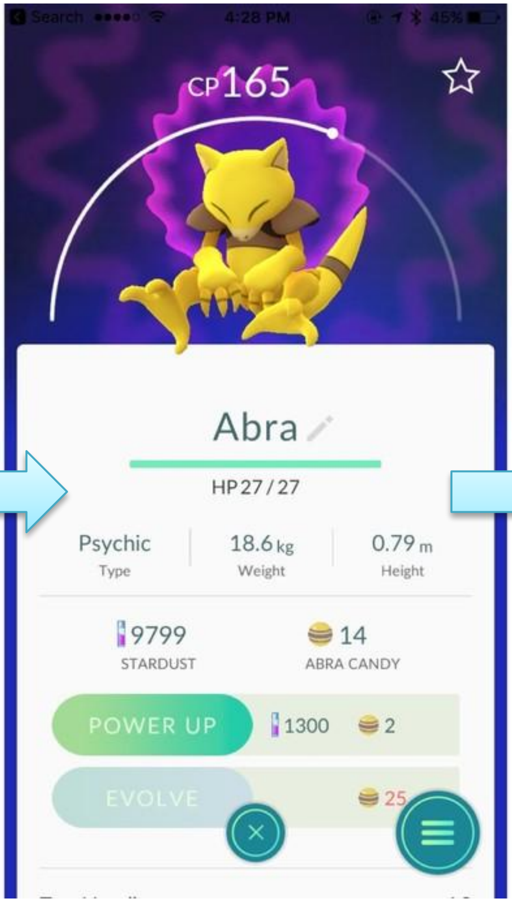
HP 10 / 10

Psychic	17.56 kg	0.85 m
Type	Weight	Height

9799 STARDUST — 14 ABRA CANDY

POWER UP 200 1

EVOLVE 25



CP165

Abra

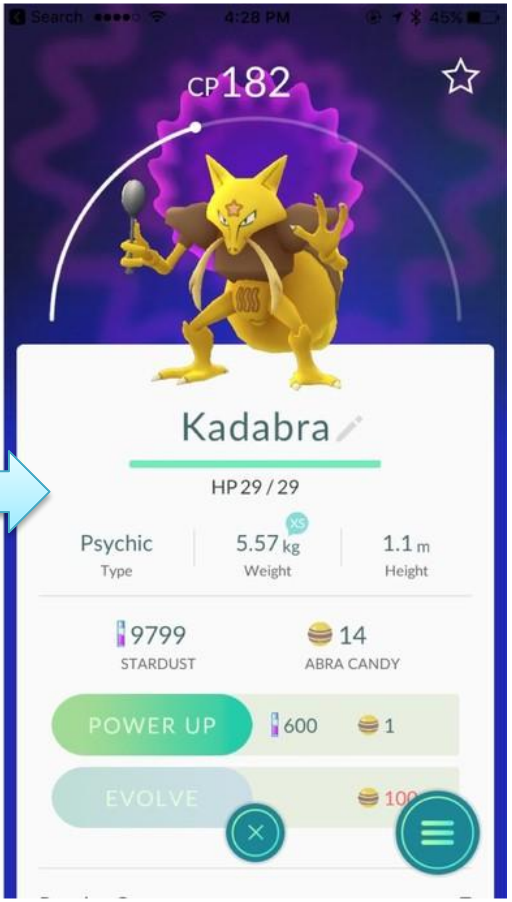
HP 27 / 27

Psychic	18.6 kg	0.79 m
Type	Weight	Height

9799 STARDUST — 14 ABRA CANDY

POWER UP 1300 2

EVOLVE 25



CP182

Kadabra

HP 29 / 29

Psychic	5.57 kg XS	1.1 m
Type	Weight	Height

9799 STARDUST — 14 ABRA CANDY

POWER UP 600 1

EVOLVE 100

Something Different

- We will NOT give you the MSVC .sln or Xcode project
  - It’s time for you to create your own

Part 1

- Implement all of the required functionality
  - insert()
  - extractMax()
  - changeKey()
  - deleteItem()

Something Different

- We will tell you what you need to do
  - Namely, list of tasks
- However
  - You have to decide the order
  - There may be some additional unnamed tasks you have to fill in

Something Different

- You have to learn how to NOT relying on the coursemology output
  - Instead relying on your own testing cases and judgement

Given Files

- The Heap files:
  - heap.h
  - heap.hpp
- The driver program for your tests
  - main.cpp
- Customer class, but you can ignore it for the Heap task first
  - customer.h
  - customer.cpp

Part 1 Heap Implementation (Array)

Very First Task

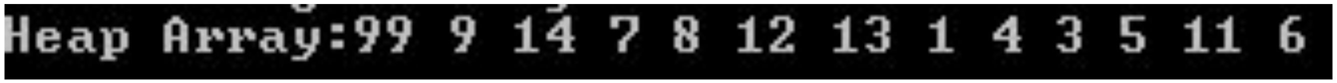
- Create your own MSVC .sln or Xcode Project

Implementing Heap

- MaxHeap
  - Parents > children
- You will implement the Heap with the array representation
  - It is **NOT** recommended to implement the heap by pointer/tree/node representation

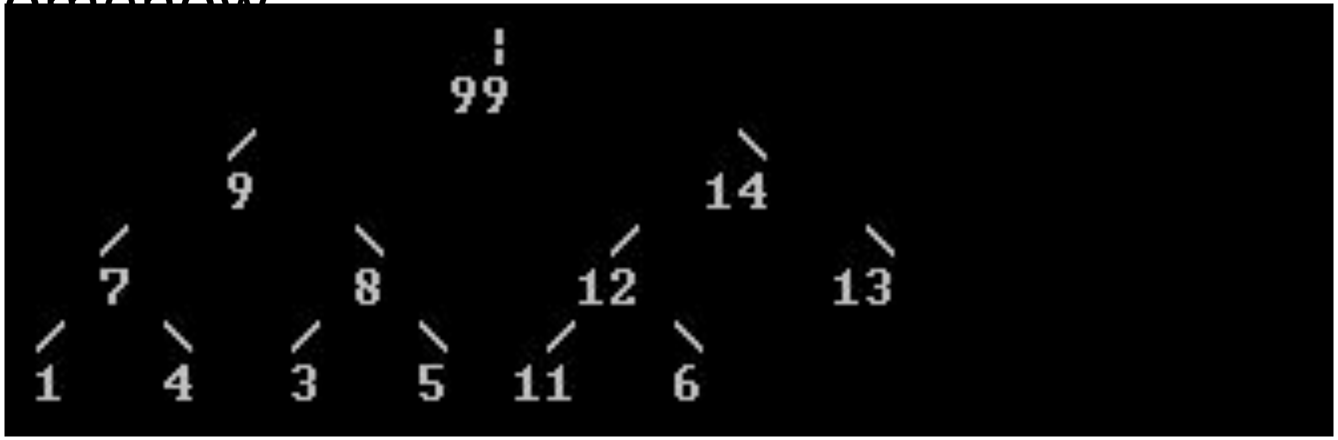
Given Functions

- `printHeapArray()`
  - print out the array that stores the heap



```
Heap Array:99 9 14 7 8 12 13 1 4 3 5 11 6
```

- `printTree()`
  - print the heap in a graphical tree form… somehow



```
                  |
                  99
            /           \
           9              14
        /     \        /     \
       7       8      12      13
     /   \   /   \   /   \
    1    4  3    5  11    6
```

Given Functions

- The function
  ```
  int Heap<T>::_lookFor(T x)
  ```
- It searches for the item x in the array and return the index of x in the heap array
- In real practice, the index of x and x should be stored in a symbol table/dictionary, thus, hash table, for fast look-up of the index of x
  - O(1) look-up time
- However, we just use a linear search here for our assignment
  - O(n) look-up time

Implementing Heap

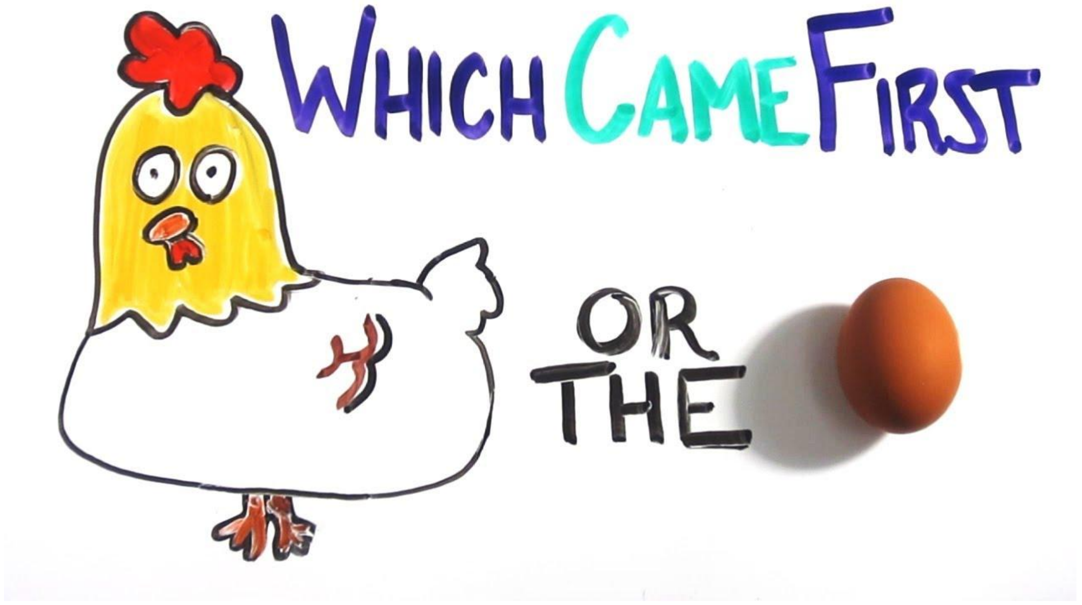
- You have to implement and submit the following functions:
  - `insert(x)`: Insert an item x into the heap
  - `extractMax()`: extract the max. item from the heap
  - `changeKey(x,y)`: decrease or increase the key of item x to y
  - `deleteItem(x)`: remove the item x

Implementing Heap

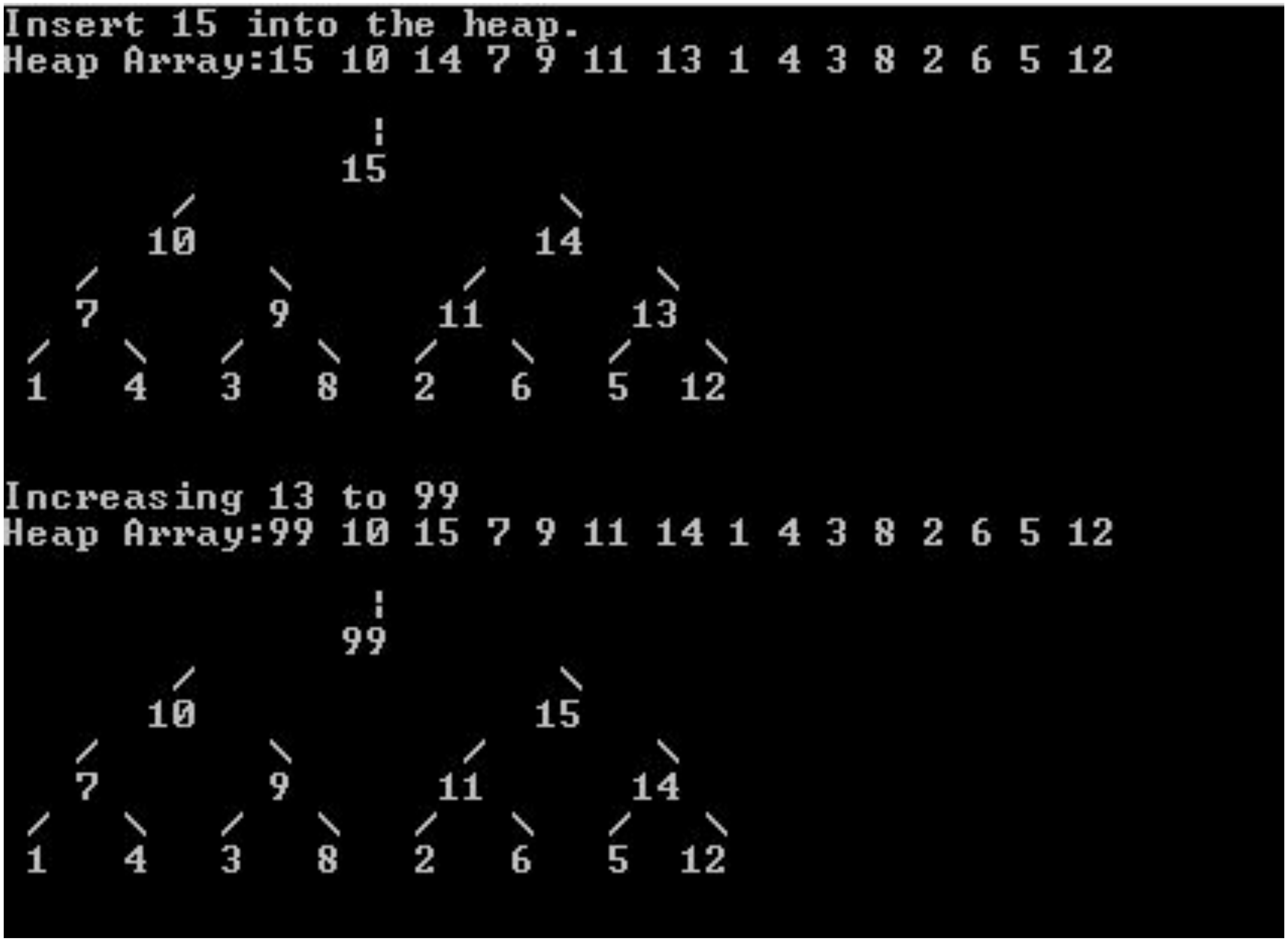
- You have to implement and submit the following functions:

  `insert(x),extractMax(),decreaseKey(x,y), increaseKey (x,y), deleteItem(x),_bubbleUp(i),_bubbleDown(i)`

- What is the order of implementation?!

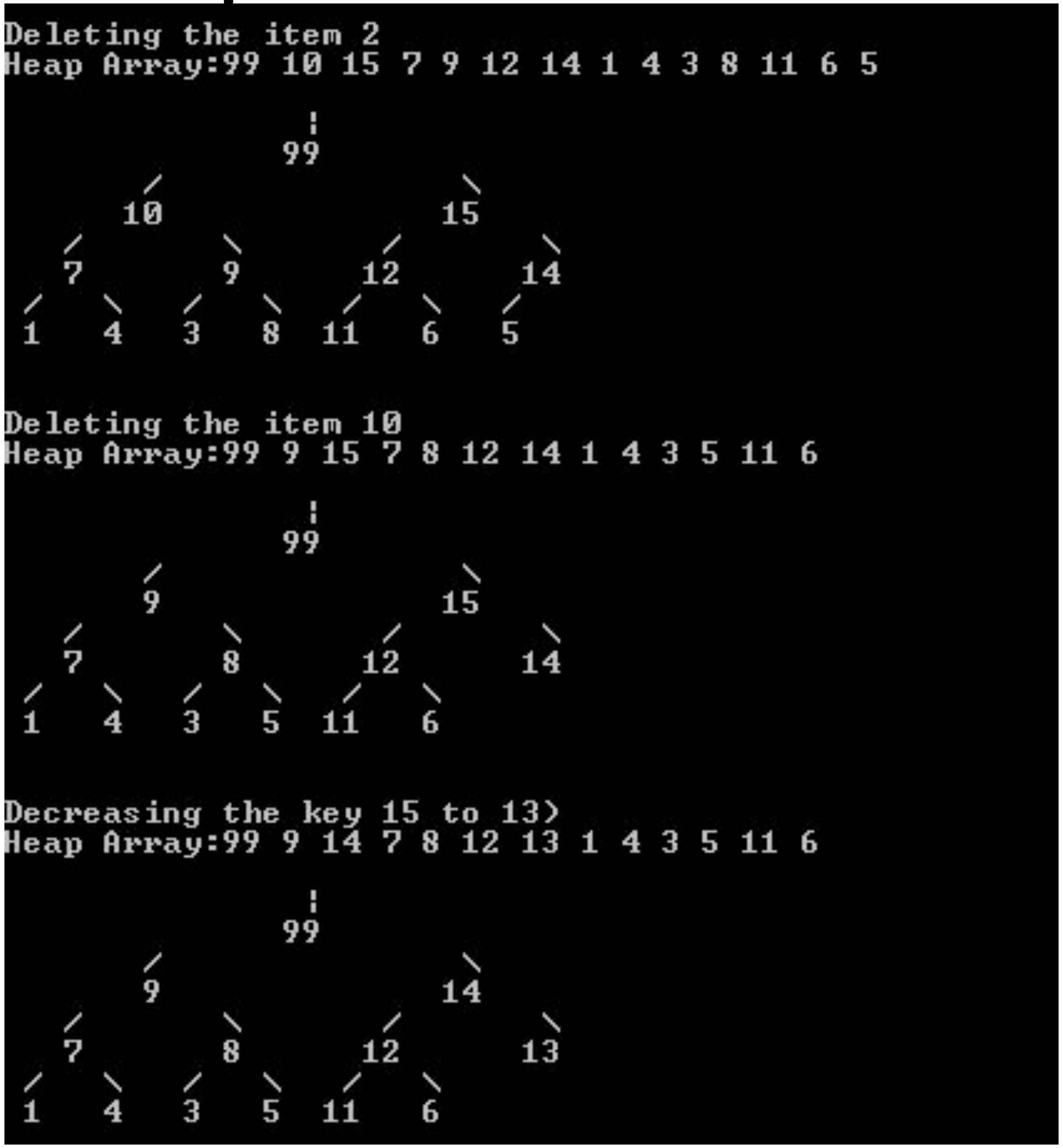


If You Implemented Correctly

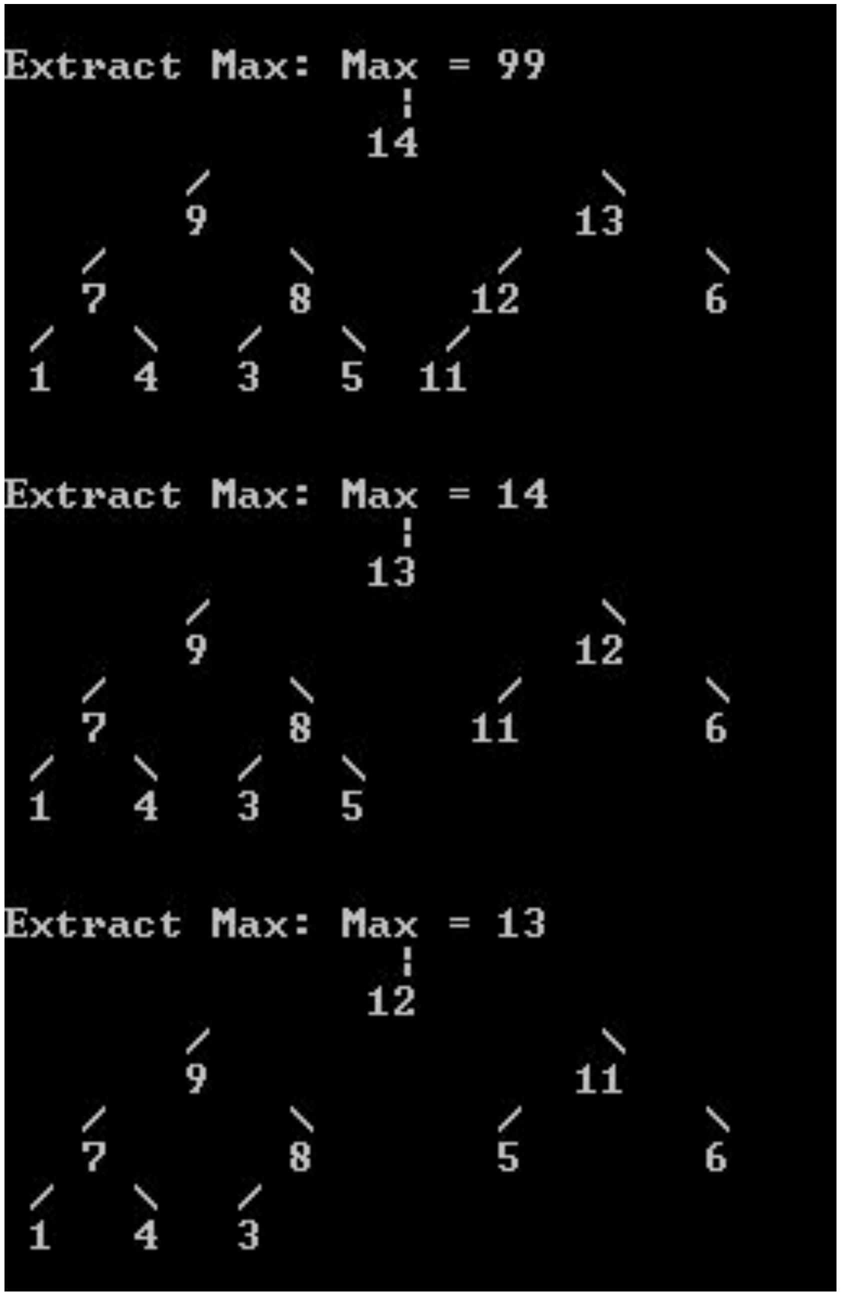


```
Insert 15 into the heap.
Heap Array:15 10 14 7 9 11 13 1 4 3 8 2 6 5 12

               |
              15
       /               \
      10              14
   /       \       /       \
   7       9      11      13
 /   \   /   \   /   \   /   \
 1   4   3   8   2   6   5  12


Increasing 13 to 99
Heap Array:99 10 15 7 9 11 14 1 4 3 8 2 6 5 12

               |
              99
       /               \
      10              15
   /       \       /       \
   7       9      11      14
 /   \   /   \   /   \   /   \
 1   4   3   8   2   6   5  12
```

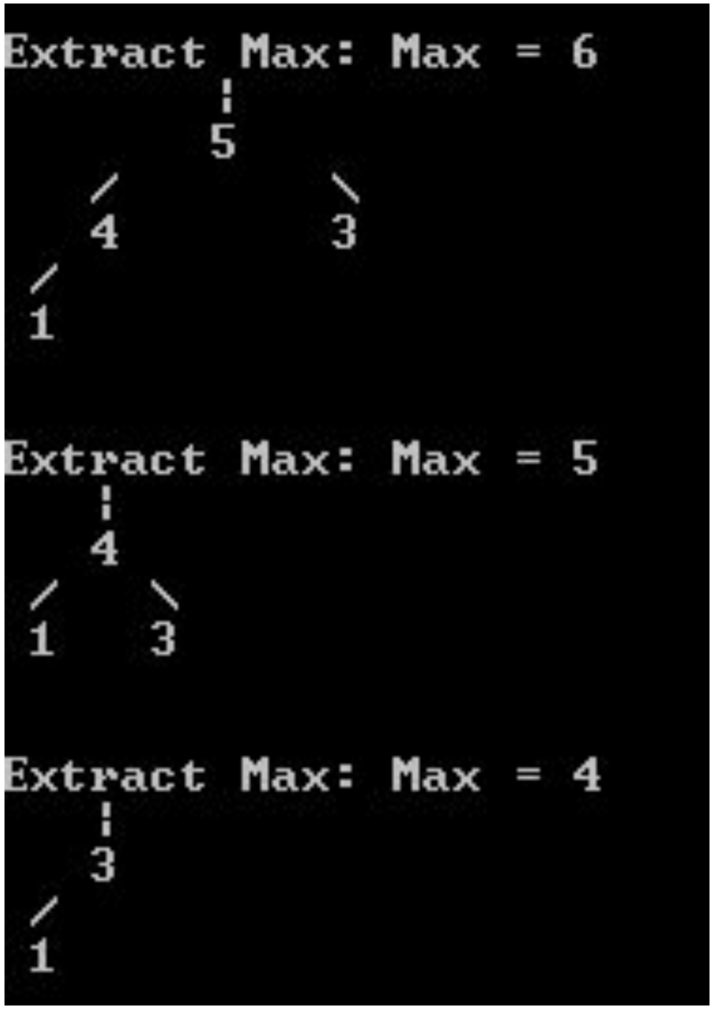
If You Implemented Correctly



```
Deleting the item 2
Heap Array:99 10 15 7 9 12 14 1 4 3 8 11 6 5

               |
              99
       /               \
      10               15
   /       \       /       \
  7         9     12       14
 / \     / \     / \     /
1   4   3   8  11   6   5

Deleting the item 10
Heap Array:99 9 15 7 8 12 14 1 4 3 5 11 6

               |
              99
       /               \
       9               15
   /       \       /       \
  7         8     12       14
 / \     / \     / \
1   4   3   5  11   6

Decreasing the key 15 to 13)
Heap Array:99 9 14 7 8 12 13 1 4 3 5 11 6

               |
              99
       /               \
       9               14
   /       \       /       \
  7         8     12       13
 / \     / \     / \
1   4   3   5  11   6
```



```
Extract Max: Max = 99
               |
              14
       /               \
       9              13
   /       \       /       \
   7       8      12       6
 /   \   /   \   /
 1   4   3   5  11

Extract Max: Max = 14
               |
              13
       /               \
       9              12
   /       \       /       \
   7       8      11       6
 /   \   /   \
 1   4   3   5

Extract Max: Max = 13
               |
              12
       /               \
       9              11
   /       \       /       \
   7       8       5       6
 /   \   /
 1   4   3
```



```
Extract Max: Max = 6
       |
       5
   /       \
   4       3
 /
 1

Extract Max: Max = 5
   |
   4
 /   \
 1   3

Extract Max: Max = 4
   |
   3
 /
 1
```

Heap Implementation Tips

- Implement a max heap (parent > children)
- Have freedom to implement any way you want so long as it passes our tests
  - … but we recommend using an array implementation
  - … of course, you cannot use any STL or ready made libraries
- Add any member variables and functions you need
- Add any test cases to ensure correctness
- See lecture slides for how a Max Heap works

Assignment Tips (both #3 & #4)

- YOUR CODE MUST BE READABLE!
  - There will be marks for code quality in both assignments. It doesn’t have to be industry quality code, but do make sure we can read your code, or you will lose marks.
- DON’T CODE USING COURSEMOLOGY!
  - Not only is this inefficient, but you’re not learning to use the tools properly.
  - You will NOT have access to Coursemology for PE!
  - The private tests in Coursemology are meant to assess the correctness of your code. Add unit tests to your own code, otherwise you will fail PE.

Queue Simulation

Second Part

Second Part: Queueing Simulation

- Queueing theory is the mathematical study of waiting lines, or queues. A queueing model is constructed so that queue lengths and waiting time can be predicted. Queueing theory is generally considered a branch of operations research because the results are often used when making business decisions about the resources needed to provide a service.

The Science of Lines

What's really happening at checkout

A shopper can use this **formula,** by John D.C. Little, to determine expected wait time: Average wait time = average number of people in line divided by their arrival rate.

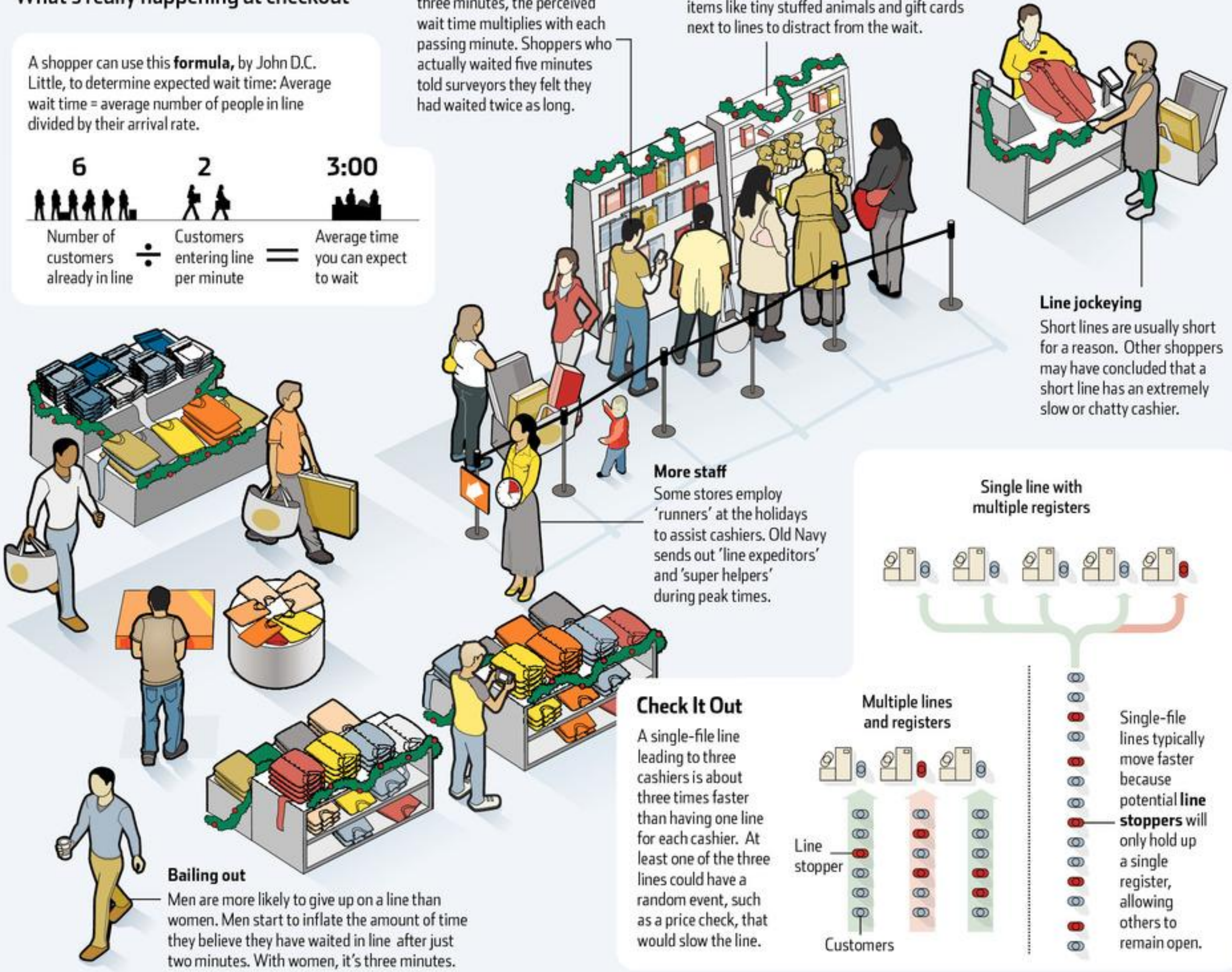
Number of customers already in line ÷ Customers entering line per minute = Average time you can expect to wait

Clock watching

Once a wait lasts longer than three minutes, the perceived wait time multiplies with each passing minute. Shoppers who actually waited five minutes told surveyors they felt they had waited twice as long.

Impulse buying

Mall retailers are copying grocery stores with items like tiny stuffed animals and gift cards next to lines to distract from the wait.



Line jockeying

Short lines are usually short for a reason. Other shoppers may have concluded that a short line has an extremely slow or chatty cashier.

More staff

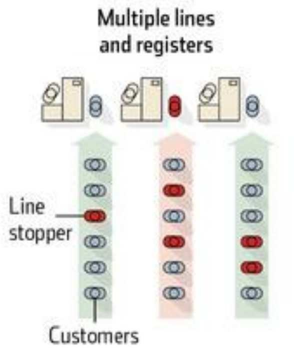
Some stores employ 'runners' at the holidays to assist cashiers. Old Navy sends out 'line expeditors' and 'super helpers' during peak times.

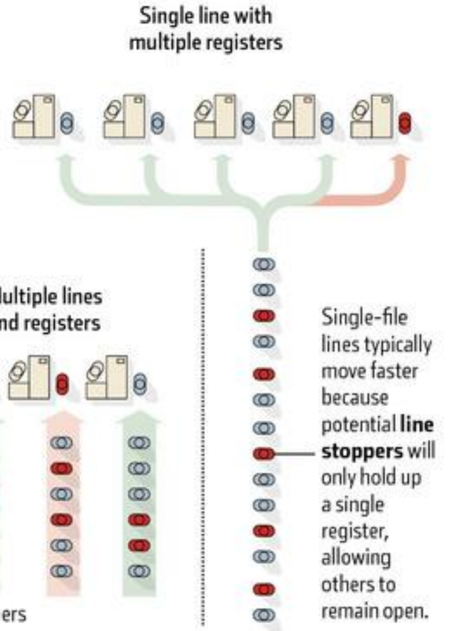
Bailing out

Men are more likely to give up on a line than women. Men start to inflate the amount of time they believe they have waited in line after just two minutes. With women, it's three minutes.

Check It Out

A single-file line leading to three cashiers is about three times faster than having one line for each cashier. At least one of the three lines could have a random event, such as a price check, that would slow the line.





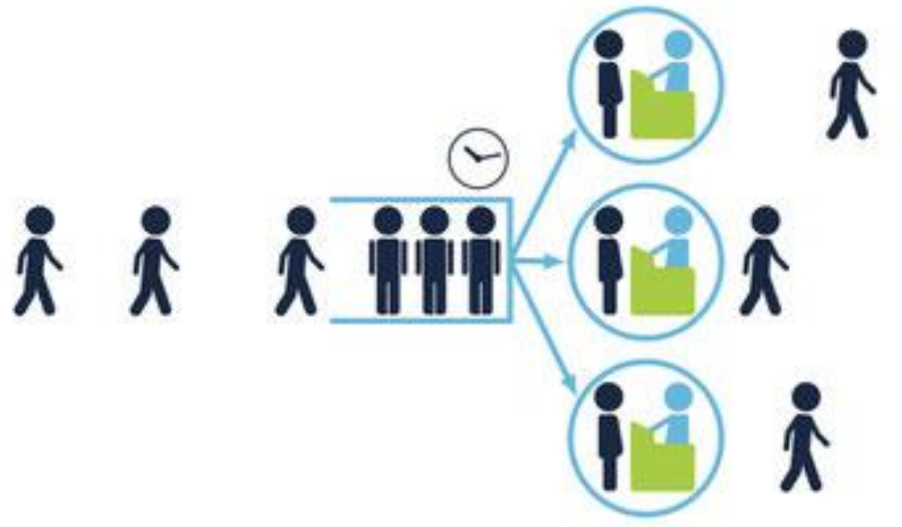
Single-file lines typically move faster because potential **line stoppers** will only hold up a single register, allowing others to remain open.

Source: WSJ reporting

Mike Sudal/The Wall Street Journal

Model

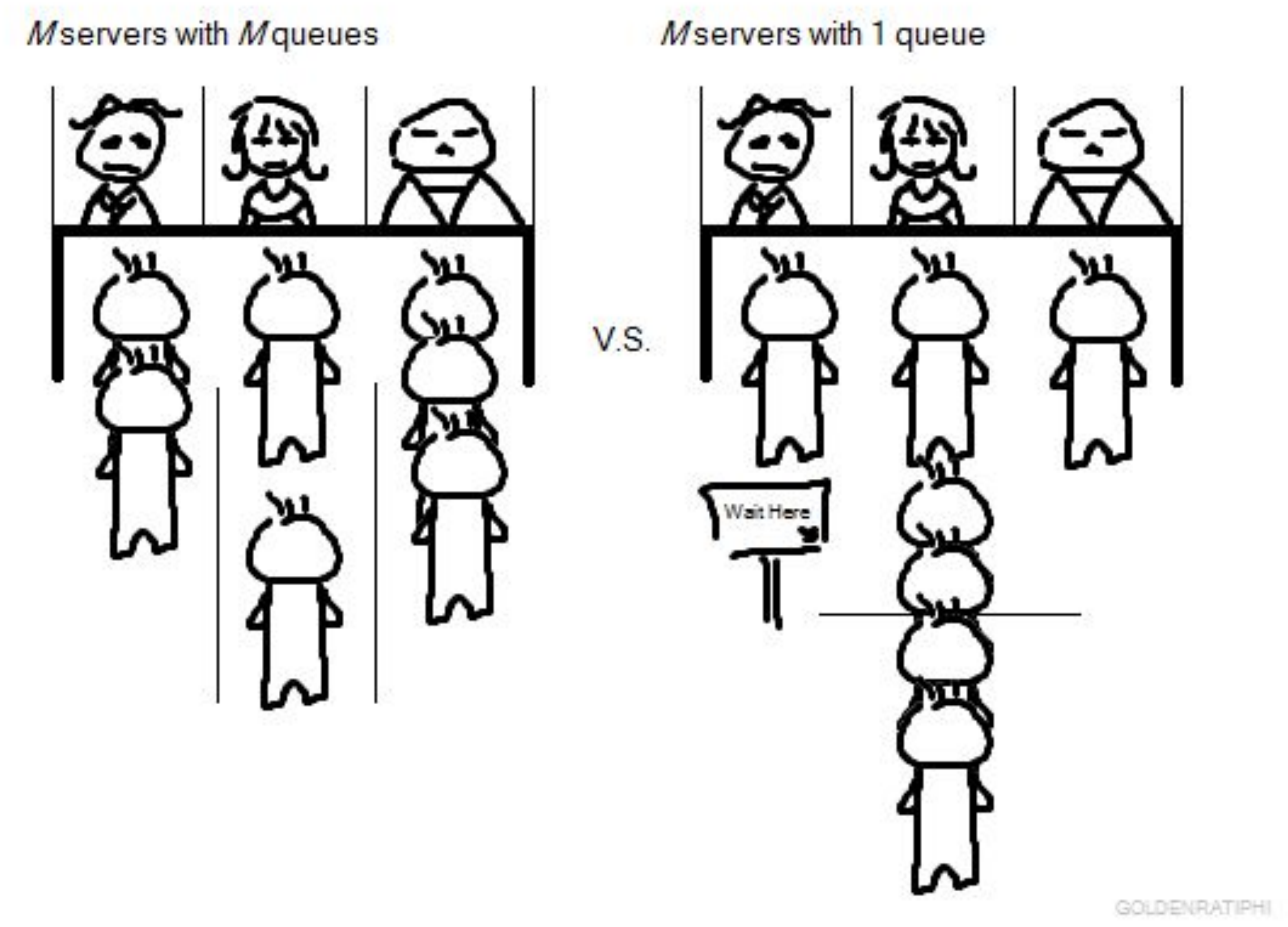
- Customers
  - Arrival time
  - Processing time
    - Time need to be servec
- Queue
  - Where customer wait, can be prioritized
- Server
  - Customers get served and leave



Questions

- Waiting time definition:
  - Time between arrival and dequeue (i.e. beginning of processing not finished processing)
- About waiting time
  - **Average WT**
  - Max WT
  - Min WT
  - or other WT
- For all customers

Single/Multiple Queue(s) with Multiple Servers



What does the bank do?

Cash in 5 minutes
for the things that matter.
只需五分钟，您就能负担得起生活中的重要所需！
1
2
3
4
5

Queue with "Priority"

YOU HAVE ELEVEN ITEMS. DROP THE ORANGE JUICE AND COME OUT WITH YOUR HANDS UP.
EXPRESS LANE 10 ITEMS OR LESS
TCOS

Tasks

- You will be given code that generates customer data for you given some parameters.
- You are to write a queuing simulator using only the code from this class to test different scenarios. The simulator will output average, min, max, and total wait times.

Questions

- If there are m servers, is it better to have m separate queues (one for each server) or a single queue that all servers dequeue from?
- If the customers have priorities assigned to them, how does that impact wait time? Is there an assignment of priorities that minimizes average wait time?

Notice

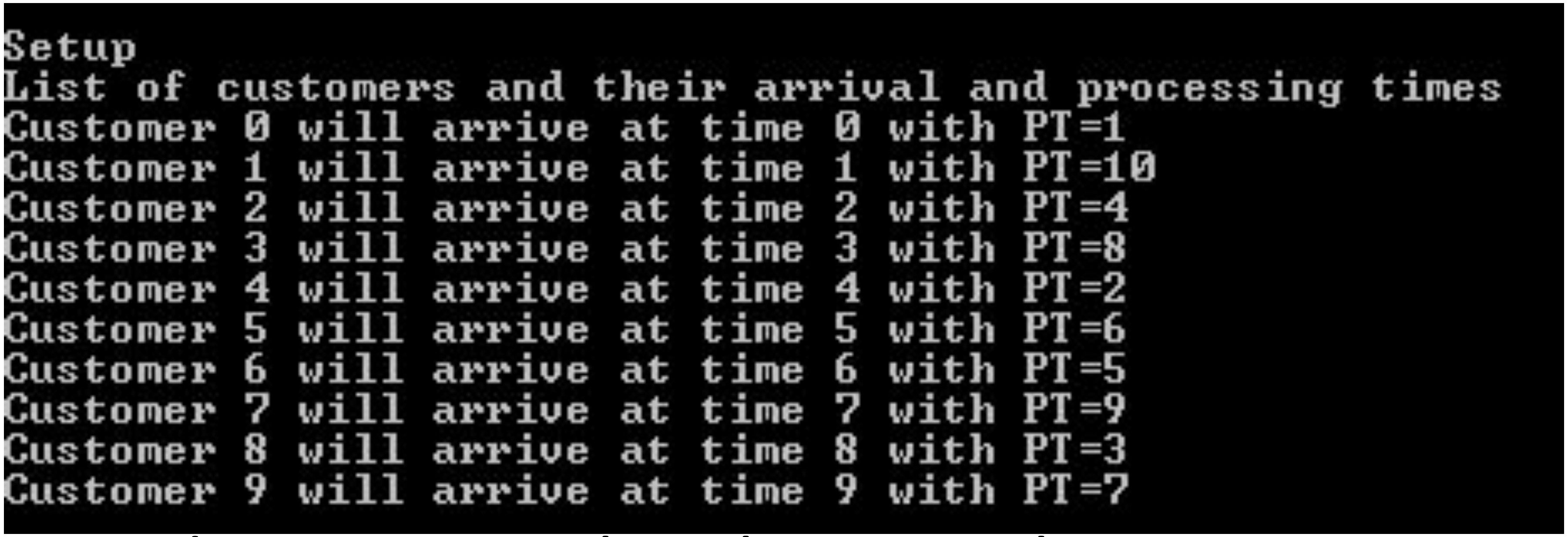
- You are not given the simulation code, nor even a skeleton for it. That is for you to write.
- Treat this like a mini-project. There are many decisions to make. Your goal isn’t just to write the code, but to answer the questions.

Submission

- Part 1
  - Heap implementation code (heap.hpp) only
- Part 2
  - Simulation code
  - Answers to questions about queueing theory

Part 2: Queue Simulation

- First, the code customerTest() will generate 10 customers for you in customer.cpp



```
Setup
List of customers and their arrival and processing times
Customer 0 will arrive at time 0 with PT=1
Customer 1 will arrive at time 1 with PT=10
Customer 2 will arrive at time 2 with PT=4
Customer 3 will arrive at time 3 with PT=8
Customer 4 will arrive at time 4 with PT=2
Customer 5 will arrive at time 5 with PT=6
Customer 6 will arrive at time 6 with PT=5
Customer 7 will arrive at time 7 with PT=9
Customer 8 will arrive at time 8 with PT=3
Customer 9 will arrive at time 9 with PT=7
```

- And we assume that there is only ONE server now

Customer Behavior

- All the customers will be stored in the array `custList[]`
- To make it simple, the customer number is equal to the time they arrived in minutes
  - E.g. Customer 4 will arrive 4 min after the store opens
- When the customer arrives, they will wait in the waiting queue if the server is serving another customer

Customer Behavior

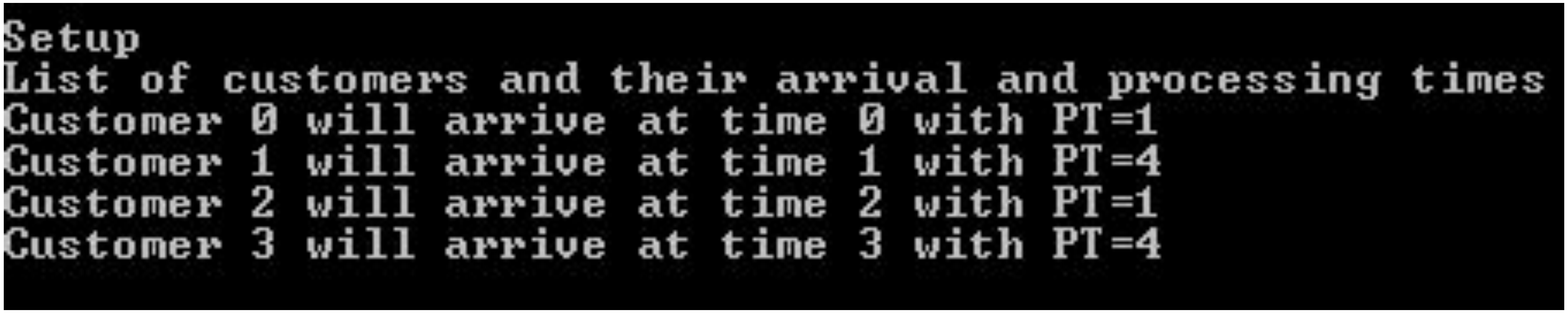
- If the server is serving no one, next customer in the queue will be served (dequeued)
- Each customer has a processing time (PT)
  - The amount of time he will occupy the server when he left the waiting queue
  - The server will not server the next one in the queue until he finish this current one

Waiting Time

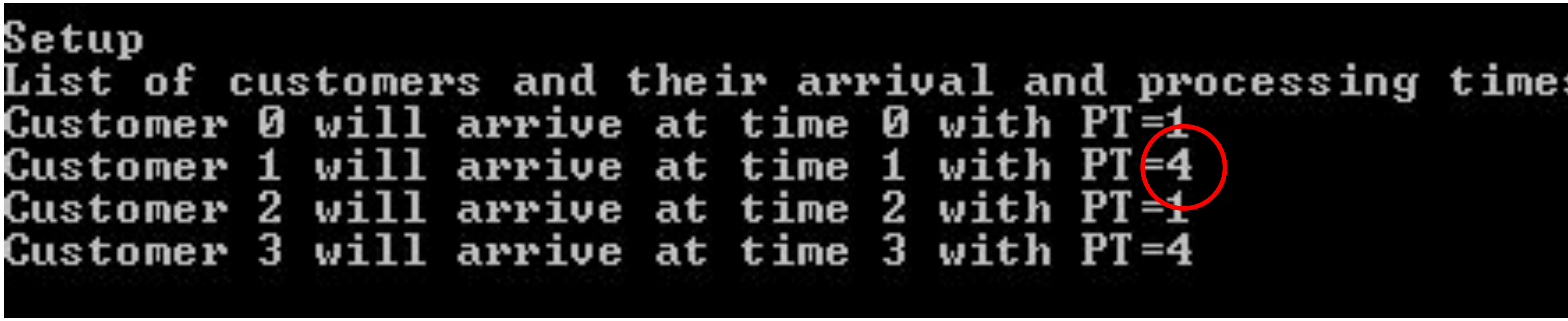
- For each customer, his waiting time (WT) is count as the difference between
  - The time he **arrives** and the time he **starts to be served** (dequeued)
  - NOT the time he finished being served

Example (Normal Queue)

- Assume that the priority of the queue is FIFO
  - Normal queue in real life
- Generate four customers (Done for you):



```
Setup
List of customers and their arrival and processing times
Customer 0 will arrive at time 0 with PT=1
Customer 1 will arrive at time 1 with PT=4
Customer 2 will arrive at time 2 with PT=1
Customer 3 will arrive at time 3 with PT=4
```

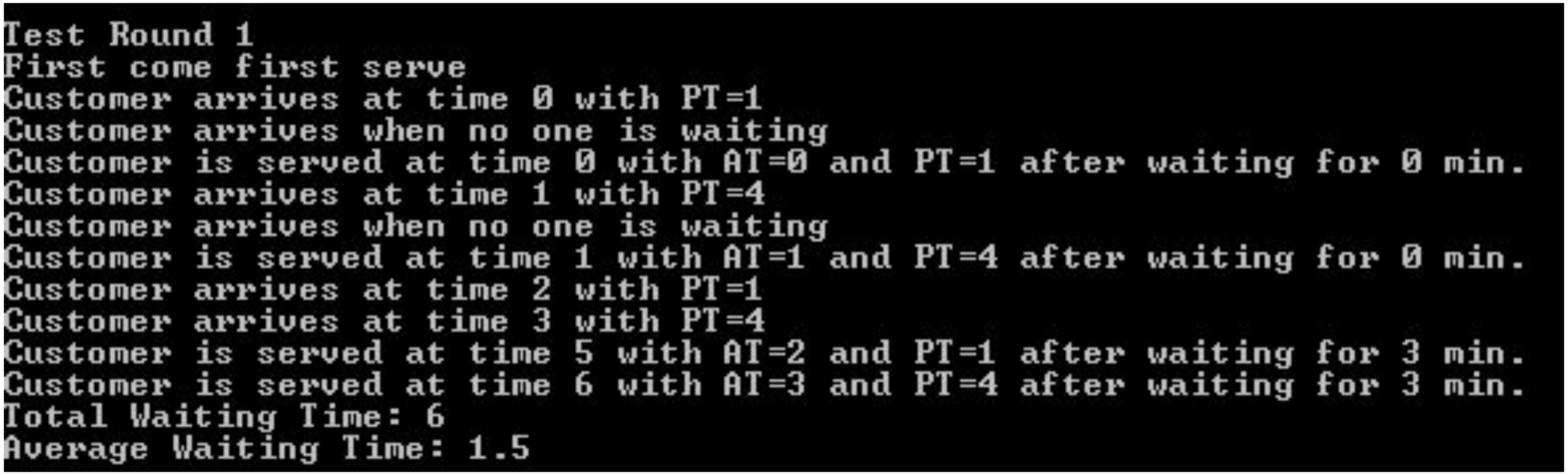


```
Setup
List of customers and their arrival and processing time
Customer 0 will arrive at time 0 with PT=1
Customer 1 will arrive at time 1 with PT=4
Customer 2 will arrive at time 2 with PT=1
Customer 3 will arrive at time 3 with PT=4
```

Time at the start of…	Customer	Server
0	C0 arrives	Serves C0 at time 0
1	C1 arrives	Serves C1 at time 1
2	C2 arrives	Busy with C1
3	C3 arrives	Busy with C1
4		Busy with C1
5		Serves C2 at time 5
6		Serves C3 at time 6

- Customer 2 waits for 5 – 2 = 3 min in the queue
- Customer 3 waits for 6 – 3 = 3 min in the queue

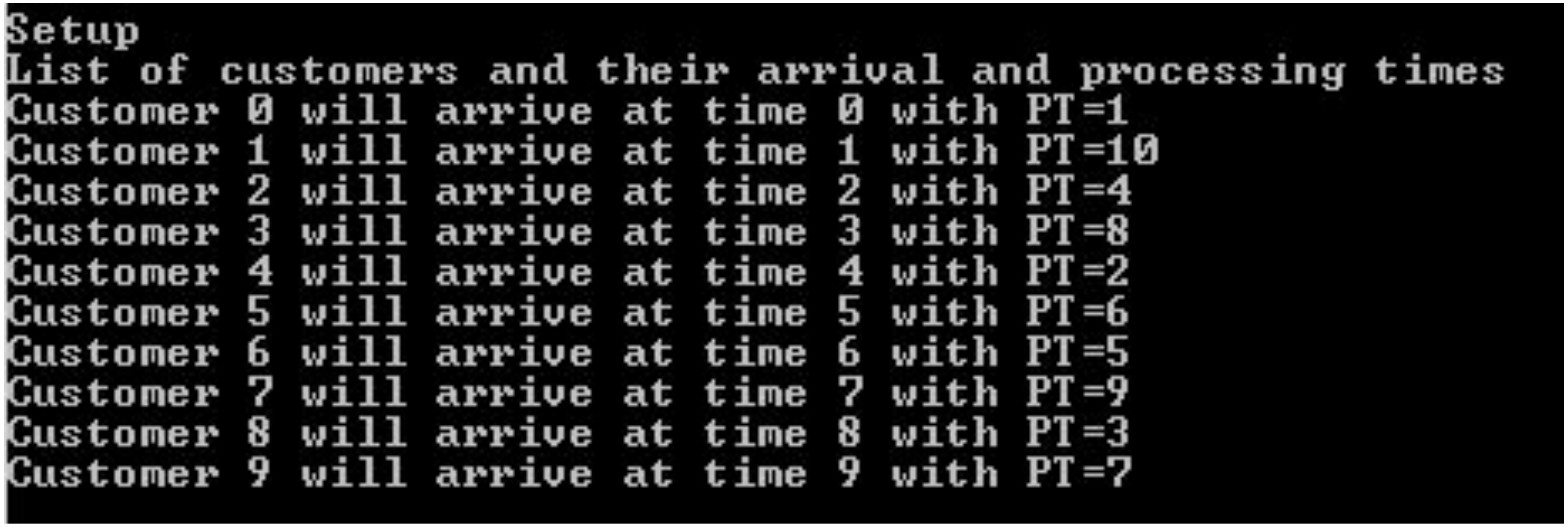
Output for FIFO



```
Test Round 1
First come first serve
Customer arrives at time 0 with PT=1
Customer arrives when no one is waiting
Customer is served at time 0 with AT=0 and PT=1 after waiting for 0 min.
Customer arrives at time 1 with PT=4
Customer arrives when no one is waiting
Customer is served at time 1 with AT=1 and PT=4 after waiting for 0 min.
Customer arrives at time 2 with PT=1
Customer arrives at time 3 with PT=4
Customer is served at time 5 with AT=2 and PT=1 after waiting for 3 min.
Customer is served at time 6 with AT=3 and PT=4 after waiting for 3 min.
Total Waiting Time: 6
Average Waiting Time: 1.5
```

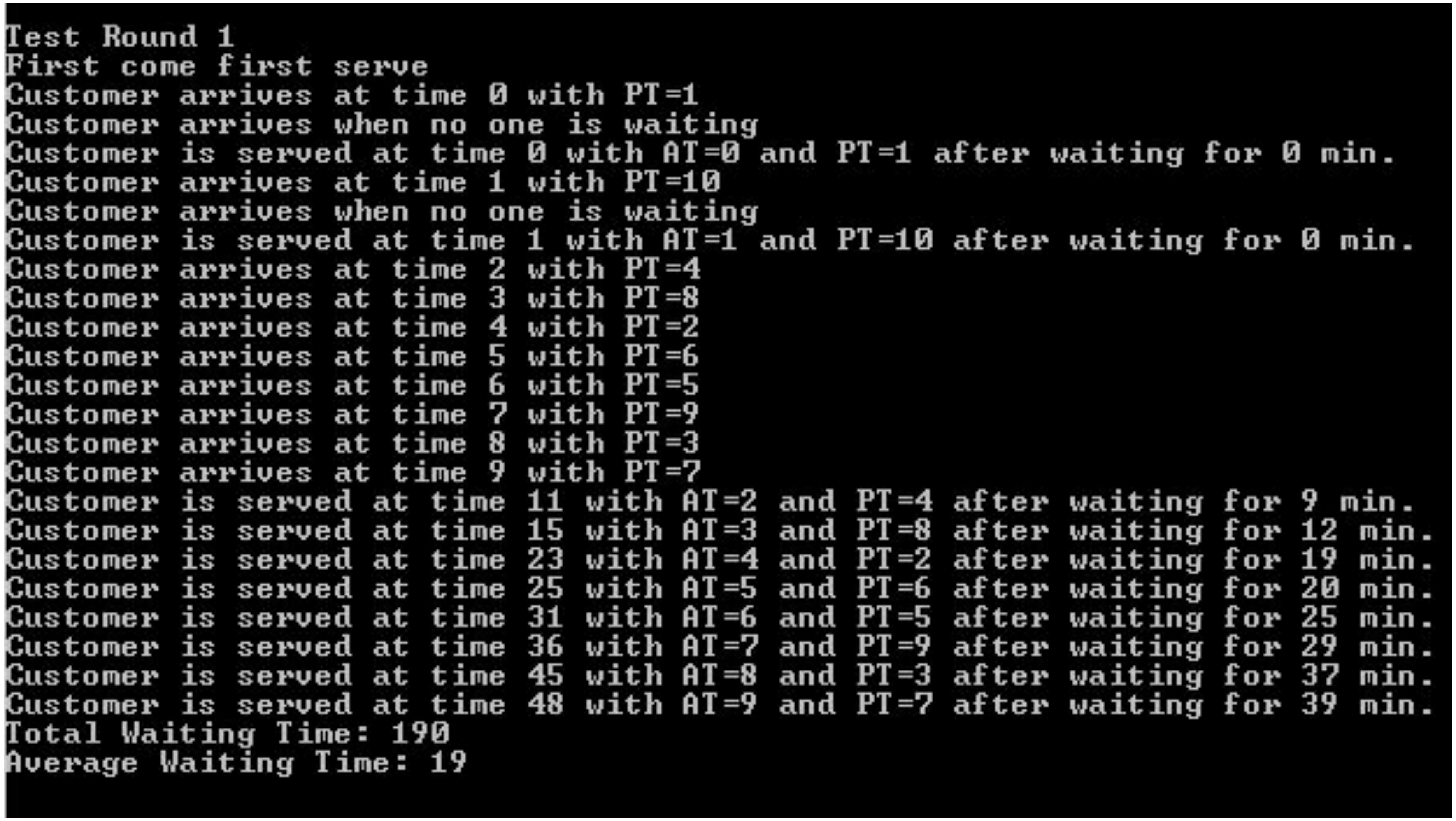
- Total WT = 6 min for all customers
- Average WT = 1.5 min for each customer

For 10 Customers (Setup)



```
Setup
List of customers and their arrival and processing times
Customer 0 will arrive at time 0 with PT=1
Customer 1 will arrive at time 1 with PT=10
Customer 2 will arrive at time 2 with PT=4
Customer 3 will arrive at time 3 with PT=8
Customer 4 will arrive at time 4 with PT=2
Customer 5 will arrive at time 5 with PT=6
Customer 6 will arrive at time 6 with PT=5
Customer 7 will arrive at time 7 with PT=9
Customer 8 will arrive at time 8 with PT=3
Customer 9 will arrive at time 9 with PT=7
```

Output for FIFO (10 Customers)

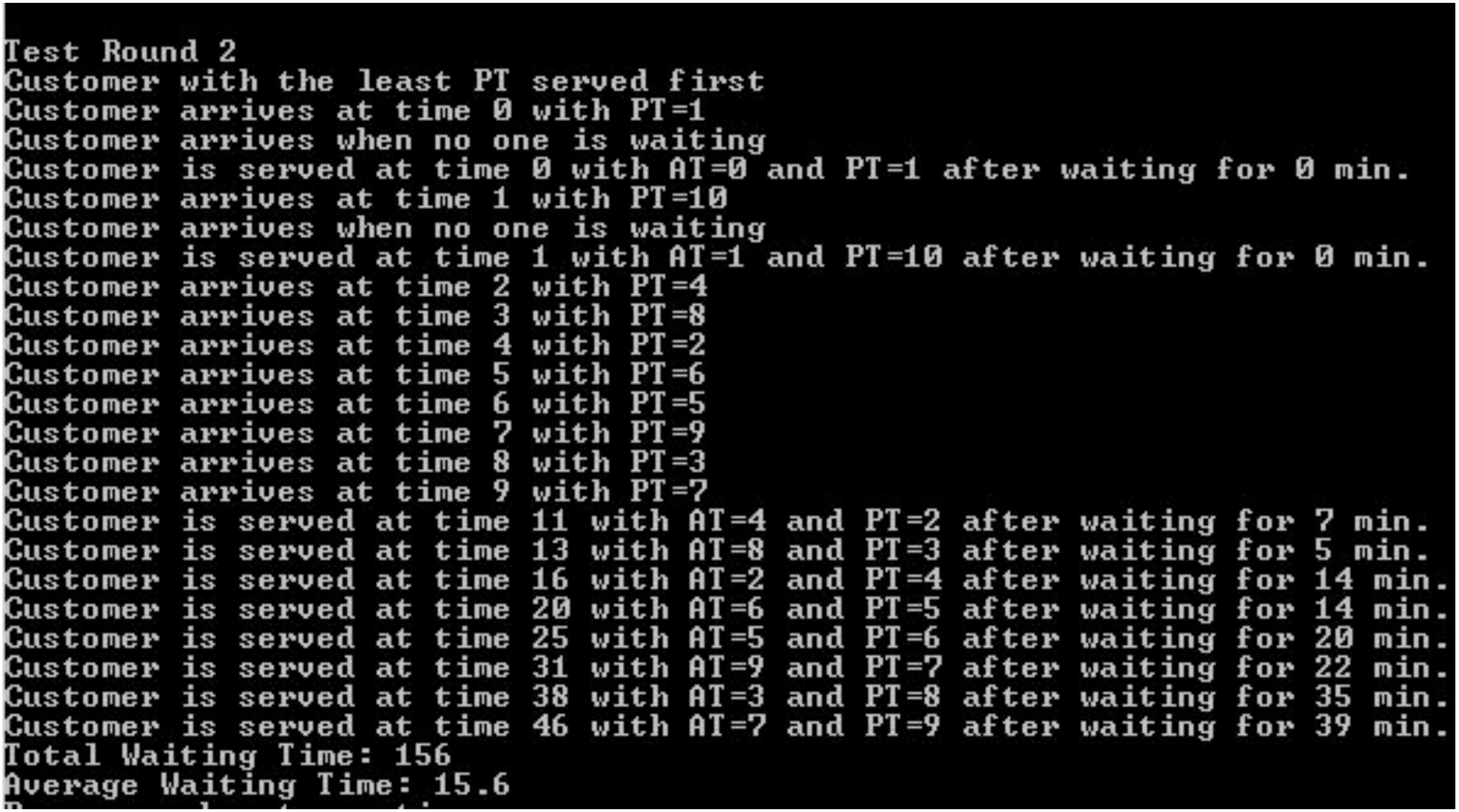


```
Test Round 1
First come first serve
Customer arrives at time 0 with PT=1
Customer arrives when no one is waiting
Customer is served at time 0 with AT=0 and PT=1 after waiting for 0 min.
Customer arrives at time 1 with PT=10
Customer arrives when no one is waiting
Customer is served at time 1 with AT=1 and PT=10 after waiting for 0 min.
Customer arrives at time 2 with PT=4
Customer arrives at time 3 with PT=8
Customer arrives at time 4 with PT=2
Customer arrives at time 5 with PT=6
Customer arrives at time 6 with PT=5
Customer arrives at time 7 with PT=9
Customer arrives at time 8 with PT=3
Customer arrives at time 9 with PT=7
Customer is served at time 11 with AT=2 and PT=4 after waiting for 9 min.
Customer is served at time 15 with AT=3 and PT=8 after waiting for 12 min.
Customer is served at time 23 with AT=4 and PT=2 after waiting for 19 min.
Customer is served at time 25 with AT=5 and PT=6 after waiting for 20 min.
Customer is served at time 31 with AT=6 and PT=5 after waiting for 25 min.
Customer is served at time 36 with AT=7 and PT=9 after waiting for 29 min.
Customer is served at time 45 with AT=8 and PT=3 after waiting for 37 min.
Customer is served at time 48 with AT=9 and PT=7 after waiting for 39 min.
Total Waiting Time: 190
Average Waiting Time: 19
```

However

- If we change the queue priority
  - It will not be First Come First Served
- Miraculously, there is a smart scanner that can detect the processing time for the customer in the queue
- **The server will serve the customer with the LEAST processing time among the people in the queue FIRST**

Output For Least PT (10 Customers)



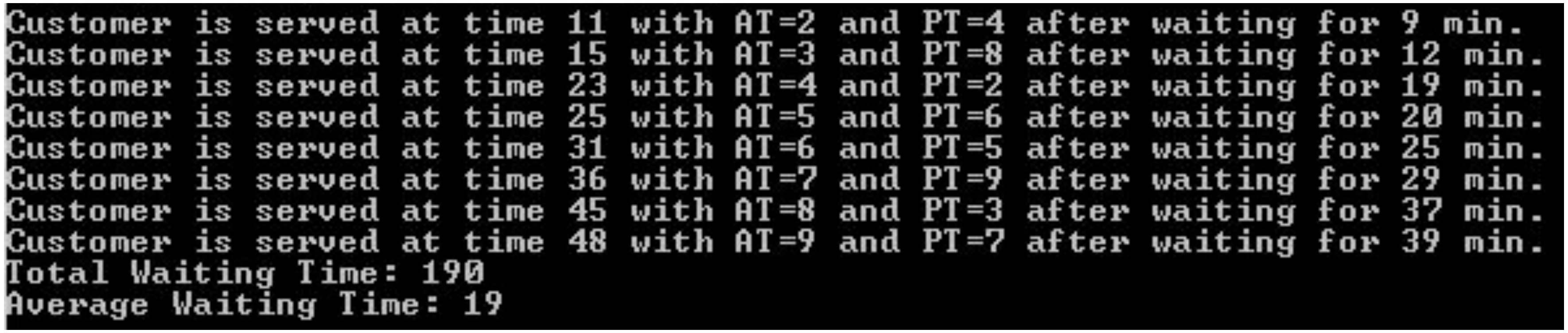
```
Test Round 2
Customer with the least PT served first
Customer arrives at time 0 with PT=1
Customer arrives when no one is waiting
Customer is served at time 0 with AT=0 and PT=1 after waiting for 0 min.
Customer arrives at time 1 with PT=10
Customer arrives when no one is waiting
Customer is served at time 1 with AT=1 and PT=10 after waiting for 0 min.
Customer arrives at time 2 with PT=4
Customer arrives at time 3 with PT=8
Customer arrives at time 4 with PT=2
Customer arrives at time 5 with PT=6
Customer arrives at time 6 with PT=5
Customer arrives at time 7 with PT=9
Customer arrives at time 8 with PT=3
Customer arrives at time 9 with PT=7
Customer is served at time 11 with AT=4 and PT=2 after waiting for 7 min.
Customer is served at time 13 with AT=8 and PT=3 after waiting for 5 min.
Customer is served at time 16 with AT=2 and PT=4 after waiting for 14 min.
Customer is served at time 20 with AT=6 and PT=5 after waiting for 14 min.
Customer is served at time 25 with AT=5 and PT=6 after waiting for 20 min.
Customer is served at time 31 with AT=9 and PT=7 after waiting for 22 min.
Customer is served at time 38 with AT=3 and PT=8 after waiting for 35 min.
Customer is served at time 46 with AT=7 and PT=9 after waiting for 39 min.
Total Waiting Time: 156
Average Waiting Time: 15.6
```

Conclusion?

- What is the difference in the average waiting time for two different policy of the priority of the queue?
  - FIFO vs Least PT
- Is it just coincident? Or is it always like that?

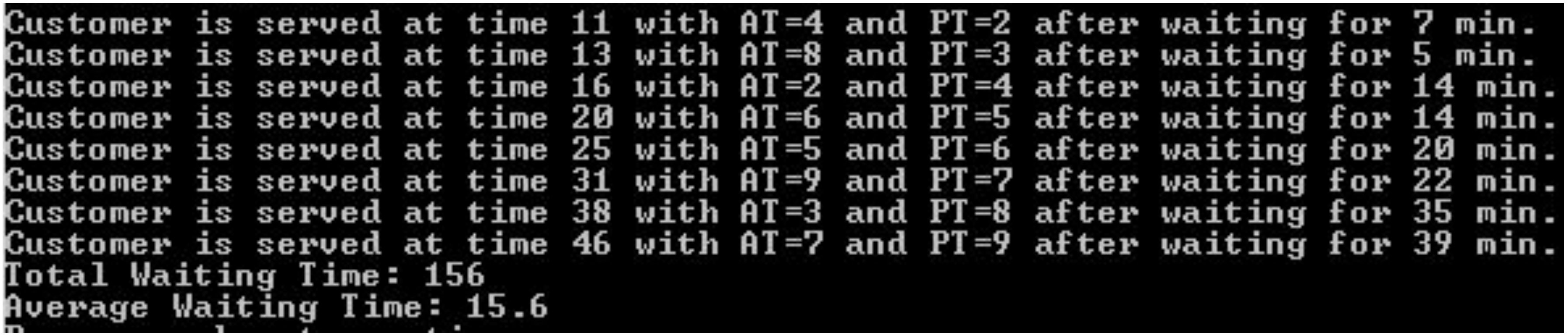
Difference

- FIFO



```
Customer is served at time 11 with AT=2 and PT=4 after waiting for 9 min.
Customer is served at time 15 with AT=3 and PT=8 after waiting for 12 min.
Customer is served at time 23 with AT=4 and PT=2 after waiting for 19 min.
Customer is served at time 25 with AT=5 and PT=6 after waiting for 20 min.
Customer is served at time 31 with AT=6 and PT=5 after waiting for 25 min.
Customer is served at time 36 with AT=7 and PT=9 after waiting for 29 min.
Customer is served at time 45 with AT=8 and PT=3 after waiting for 37 min.
Customer is served at time 48 with AT=9 and PT=7 after waiting for 39 min.
Total Waiting Time: 190
Average Waiting Time: 19
```

- Least PT first



```
Customer is served at time 11 with AT=4 and PT=2 after waiting for 7 min.
Customer is served at time 13 with AT=8 and PT=3 after waiting for 5 min.
Customer is served at time 16 with AT=2 and PT=4 after waiting for 14 min.
Customer is served at time 20 with AT=6 and PT=5 after waiting for 14 min.
Customer is served at time 25 with AT=5 and PT=6 after waiting for 20 min.
Customer is served at time 31 with AT=9 and PT=7 after waiting for 22 min.
Customer is served at time 38 with AT=3 and PT=8 after waiting for 35 min.
Customer is served at time 46 with AT=7 and PT=9 after waiting for 39 min.
Total Waiting Time: 156
Average Waiting Time: 15.6
```

Given Code

```
class Customer {
private:
    int arrival_time;
// time of arrival after the shop opened in min
    int processing_time;
// amount of time need to be processed with the customer serivce in
min

public:
    Customer () {arrival_time = 0; processing_time = 0;};
    void setAT(int t) {arrival_time = t;};
    void setPT(int t) {processing_time = t;};
    int AT() {return arrival_time;};
    int PT() {return processing_time;};

    bool operator>(const Customer& c);
    // a customer is "greater" if his time is shorter

};
```

Comparison For Heap/PQ

- If comparisonWay = 0
  - the heap be ordered by arrival times (AT)
- If comparisonWay = 1
  - the heap be ordered by processing times (PT)

```
int comparisonWay = 0; // 0 for arrival time, 1 for processing
time


bool Customer::operator>(const Customer& c) {
return comparisonWay ?
       processing_time < c.processing_time :
       arrival_time < c.arrival_time;
       // a customer is "greater" if his time is shorter
};
```

Actually NOT a good practice to use a global variable

First Part of CustomerQueueTest()

Setting up n customers for the test

```
void customerQueueTest(int n_cust) {
  int current_time = 0;
  int totalWaitingTime = 0;
  int i;
  int WT = 0; // waiting time for each customer

  Heap<Customer> queue;

  Customer* custList = new Customer[n_cust];
  cout << endl << endl << "Setup" << endl;
  cout << "List of customers and their arrival and processing times" <<
endl;
  for (i = 0; i<n_cust; i++)
  {
    custList[i].setAT(i);
    custList[i].setPT((n_cust - i) % (n_cust / 2) + 1 + (i % 2)*(n_cust /
2));
    cout << "Customer " << i << " will arrive at time "
         << custList[i].AT() << " with PT=" << custList[i].PT() << endl;
  }
  cout << endl;
```

Following on

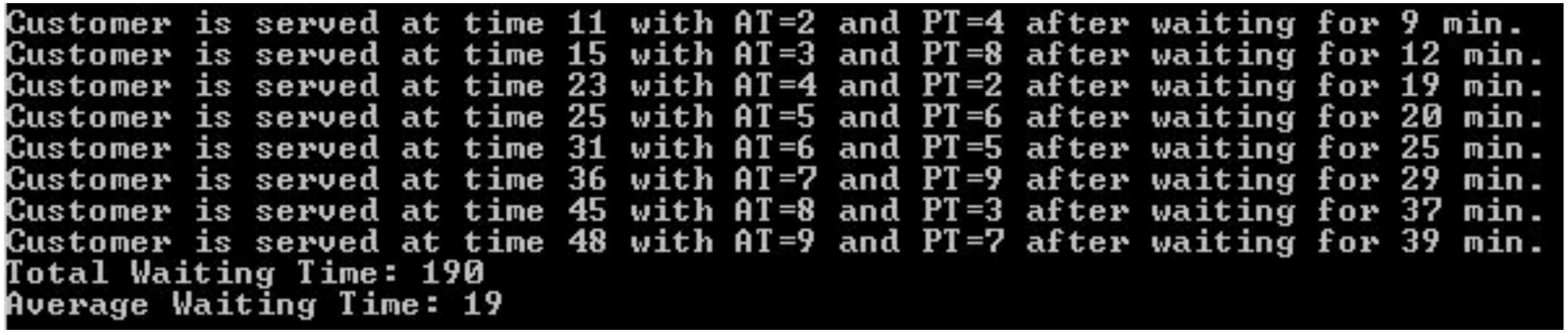
```
for (int round = 0; round<2; round++) {
  // you may need a big modification within this for-loop

  cout << endl << endl;
  cout << "Test Round " << round + 1 << endl;
  cout << (round == 0 ? "First come first serve" : "Customer with the least PT served first") << endl;
  comparisonWay = round;
  current_time = 0;
  totalWaitingTime = 0;
  queue.insert(custList[0]);

  cout << "Customer arrives at                                  h PT=" << custList[0].PT() << endl;

  while (!queue.empty()) {

    // you should change all of
    Customer c = queue.extractM
    cout << "Customer arrives when no one is waiting" << endl;
    cout << "Customer is served at time " << current_time << " with AT=" << c.AT()
         << " and PT=" << c.PT() << " after waiting for "
         << WT << " min." << endl;
}

cout << "Total Waiting Time: " << totalWaitingTime << endl;
cout << "Average Waiting Time: " << totalWaitingTime / (float)n_cust << endl;
```

You have to change all the code here!

The purpose for giving you the code here is for all those "cout" such that it will match the coursemology test case

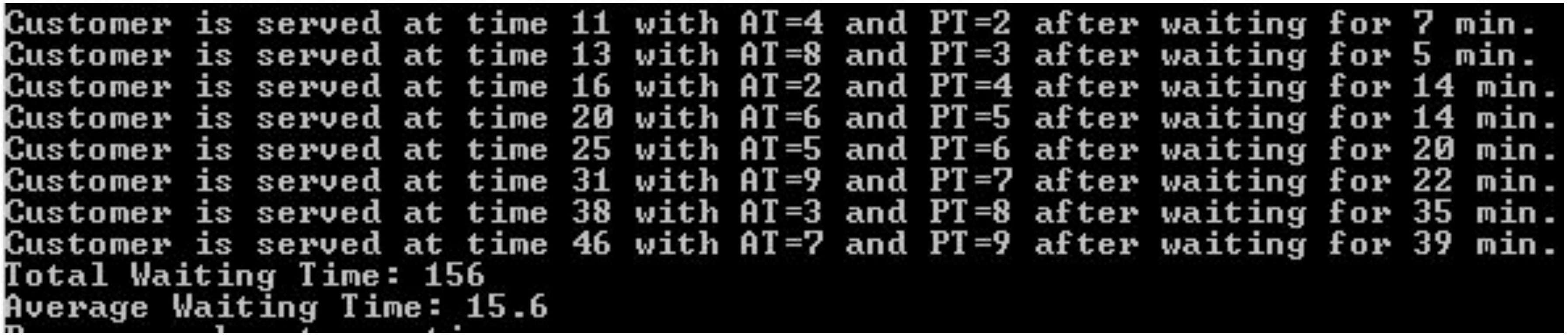
Difference

- FIFO



```
Customer is served at time 11 with AT=2 and PT=4 after waiting for 9 min.
Customer is served at time 15 with AT=3 and PT=8 after waiting for 12 min.
Customer is served at time 23 with AT=4 and PT=2 after waiting for 19 min.
Customer is served at time 25 with AT=5 and PT=6 after waiting for 20 min.
Customer is served at time 31 with AT=6 and PT=5 after waiting for 25 min.
Customer is served at time 36 with AT=7 and PT=9 after waiting for 29 min.
Customer is served at time 45 with AT=8 and PT=3 after waiting for 37 min.
Customer is served at time 48 with AT=9 and PT=7 after waiting for 39 min.
Total Waiting Time: 190
Average Waiting Time: 19
```

- Least PT first



```
Customer is served at time 11 with AT=4 and PT=2 after waiting for 7 min.
Customer is served at time 13 with AT=8 and PT=3 after waiting for 5 min.
Customer is served at time 16 with AT=2 and PT=4 after waiting for 14 min.
Customer is served at time 20 with AT=6 and PT=5 after waiting for 14 min.
Customer is served at time 25 with AT=5 and PT=6 after waiting for 20 min.
Customer is served at time 31 with AT=9 and PT=7 after waiting for 22 min.
Customer is served at time 38 with AT=3 and PT=8 after waiting for 35 min.
Customer is served at time 46 with AT=7 and PT=9 after waiting for 39 min.
Total Waiting Time: 156
Average Waiting Time: 15.6
```

And.. You may need it for the next Assignment

Real Job

- Treat the Queueing Simulation part of your assignment as a real job
  - A mini project

10 ways school is different than the working world

- If you can't do the work, you're out.
- We grade on a curve.
- …
- We hate slackers.
- You have to fight for recognition.
- We really, really, really want you to be able to write **CODE** properly
- We really, really, really want you to be able to do math.
- If you can't make money for us, we won't hire you.

https://www.cbsnews.com/news/10-ways-school-is-different-than-the-world-of-work/