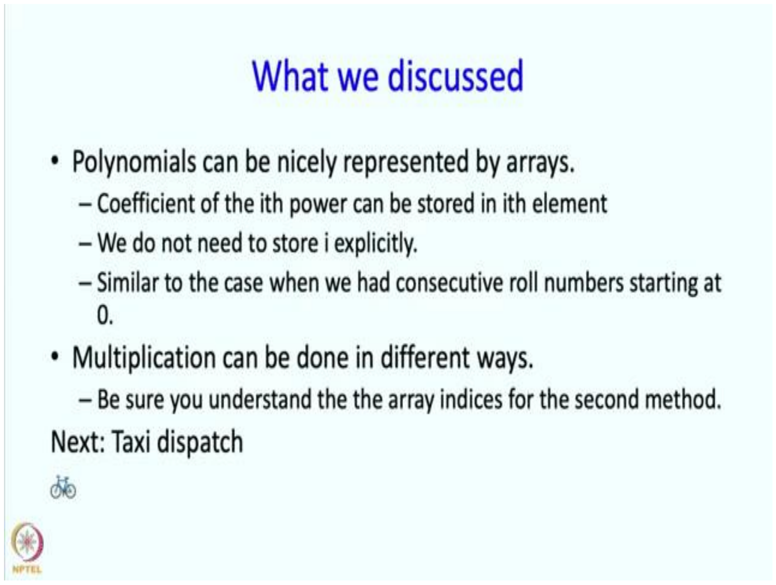
An Introduction to Programming through C++

Professor. Abhiram G. Ranade
Department of Computer Science and engineering,
Indian Institute of Technology Bombay India.

Lecture 15 Part-6
Array part- 1
Queues in dispatching taxis

(Refer Slide Time: 00:21)



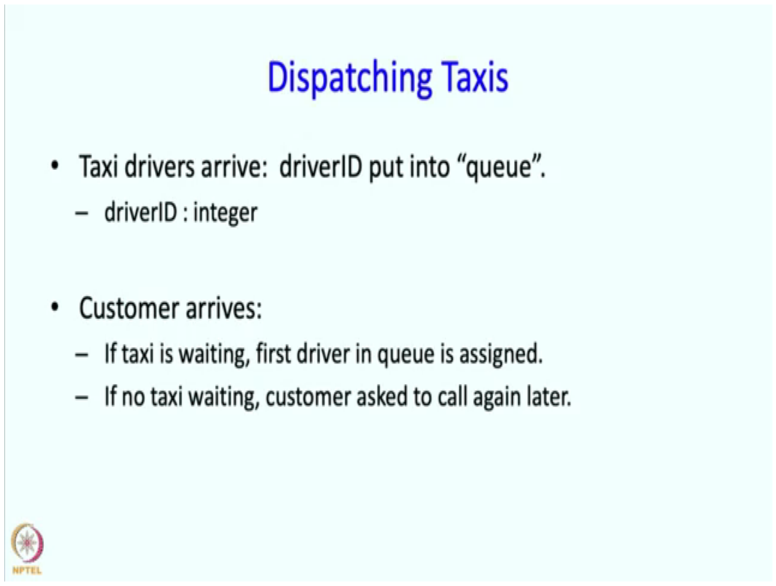
What we discussed

- Polynomials can be nicely represented by arrays.
  - Coefficient of the ith power can be stored in ith element
  - We do not need to store i explicitly.
  - Similar to the case when we had consecutive roll numbers starting at 0.
- Multiplication can be done in different ways.
  - Be sure you understand the the array indices for the second method.

Next: Taxi dispatch

Welcome back, in the last segment we discuss how Arrays can be used to represent polynomials. In this segment we are going to discuss another interesting use of Arrays in a problem called the taxi dispatch problem.

(Refer Slide Time: 00:43)



Dispatching Taxis

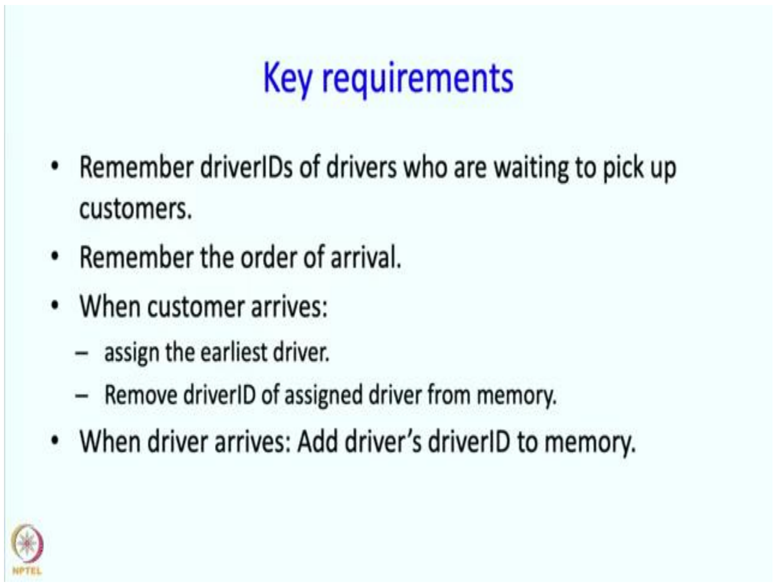
- Taxi drivers arrive: driverID put into "queue".
  - driverID : integer
- Customer arrives:
  - If taxi is waiting, first driver in queue is assigned.
  - If no taxi waiting, customer asked to call again later.

So, in this problem you are at some taxi terminal where taxi drivers arrive and wait for passengers and the idea is that the driver that arrives first must be given a passenger before anyone that comes later. So, we need to some kind of a queue over here. So, we need to somehow record who came first, who came second, and so on.

So, we are going to get the IDs. So, this will be numbers of the driver that arrive and we are going to put them in some kind of a queue. We will see what exactly how to do this? but you know what a queue is in real life. So, we need to do something like that. So, we said that the driver ID going to be an integer for simplicity but later on we can have some more complicated driver ID as well. Well maybe a customer arrives. So, if a customer arrives and if a taxi is waiting then the first driver in the queue should be assign to that customer. If no taxi is waiting then the customer is ask to call again sometime later.

So, imagine that you are the controller of this taxi dispatch centre and you need a program to keep track of everything that is going on. So, what would that program look like. So, we are ask to right this program. So, that is the problem which has been given to us.

(Refer Slide Time: 02:21)

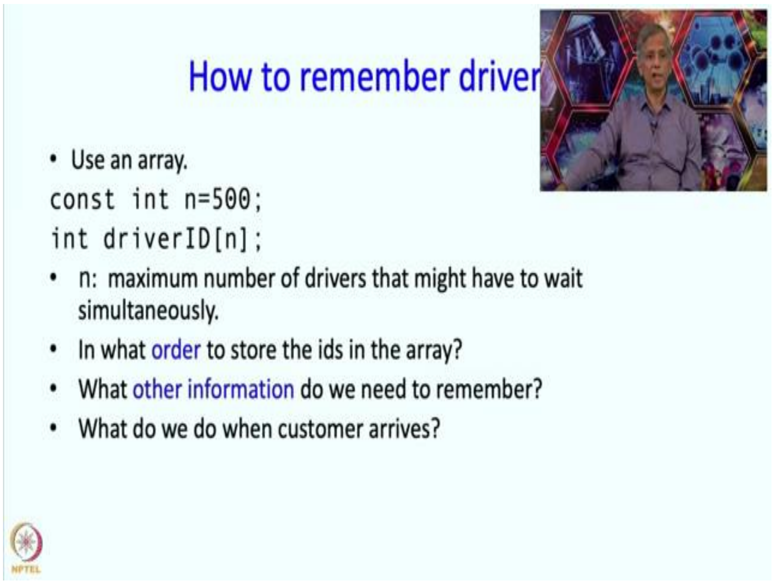


Key requirements

- Remember driverIDs of drivers who are waiting to pick up customers.
- Remember the order of arrival.
- When customer arrives:
  - assign the earliest driver.
  - Remove driverID of assigned driver from memory.
- When driver arrives: Add driver's driverID to memory.

So, what are the requirements? Well you need to remember the IDs of the drivers who are waiting to pick up customer that is one requirement and you also need to remember the order in which they arrive and when the customer arrives you assign the earliest driver. And when you assign the earliest driver you should remove the driver ID of the assigned driver from your memory, that is basically it. When driver arrives you add the driver ID to memory but you kind a put it last. So, how do you remember the driver ID's. So, if you have to remember a larger number of numbers of the same types, of course you have to use an Array. So, there is no escaping from that. So, we will use an Array.

(Refer Slide Time: 03:09)



How to remember driver

- Use an array.

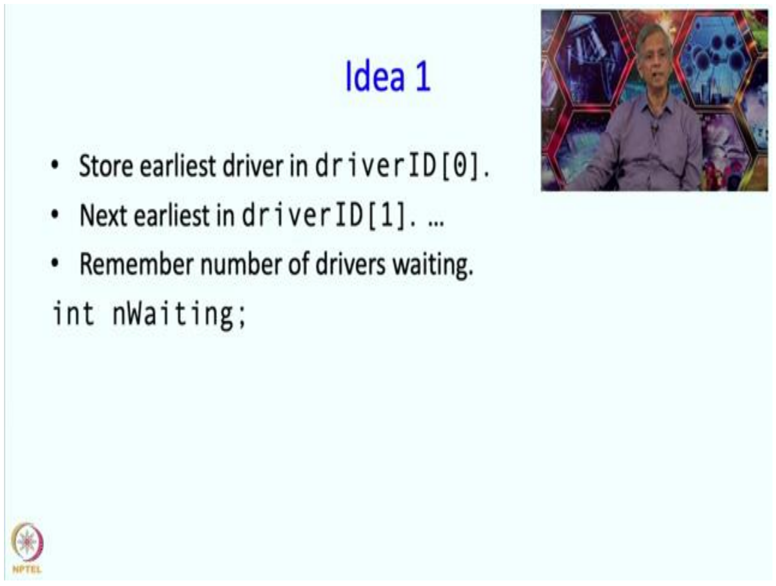
```
const int n=500;
int driverID[n];
```

- n: maximum number of drivers that might have to wait simultaneously.
- In what order to store the ids in the array?
- What other information do we need to remember?
- What do we do when customer arrives?

So, let us say we are going to remember some 500 driver ID's. So, we are going to use a name instead of the number 500 because the name tells us what it is that this number is representing. So, I could have made this name more informative I could have called it say ndrivers, but let me just keep it simple let me just keep it compact because otherwise my slides became too large. So, I am going to define an array with 500 elements. So, this array is going to called driverID.

So, if more than 500 drivers happened to be waiting then we will have a problem. So, we might have to tell the driver that look please go back and come again later because we do not have space to record your ID. Ideally this should not happen. But let us say just we have to take care of every possibility when we write a program. Now the important question is, in what order should we store the IDs in our Array. Also, besides the ID is there any other information that we need to remember and we need to say - what should we do when the customer arrives? what should we do when a driver arrives? So, these are question we need to answer.

(Refer Slide Time: 04:35)



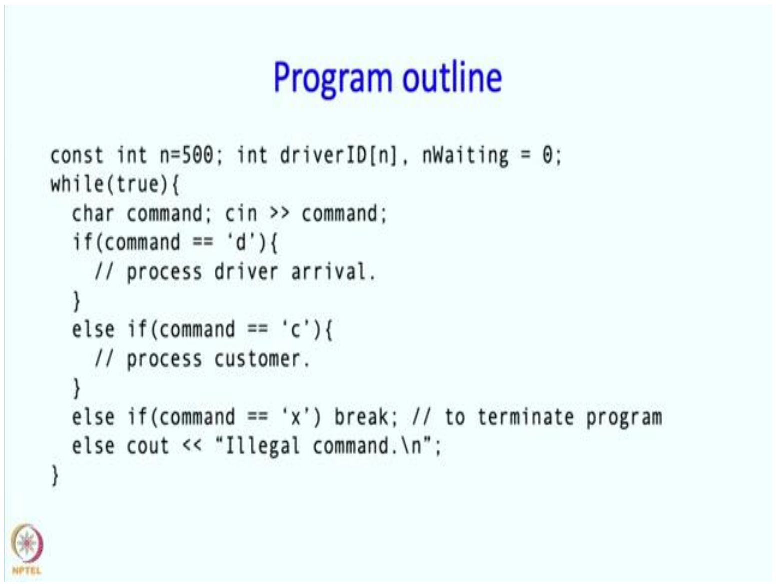
Idea 1

- Store earliest driver in `driverID[0]`.
- Next earliest in `driverID[1]`. ...
- Remember number of drivers waiting.

```
int nWaiting;
```

So, here is the first idea for answering these questions. So, the earliest driver who has not yet been served, let us say we store in the driverID[0]. The next earliest driverID[1] and so on and we also remember, how many drivers are waiting? So, for that lets say we use a variable called nWaiting. So, that is basically it we have pretty much decided what variables we need at this point well these are sort of the important variables. Later on, we may need to have some local small variables which will be needed for a short while, but really the important information as far as this program is concern it is going to be stored in this array driverID and this variable nWaiting.

(Refer Slide Time: 05:33)



Program outline

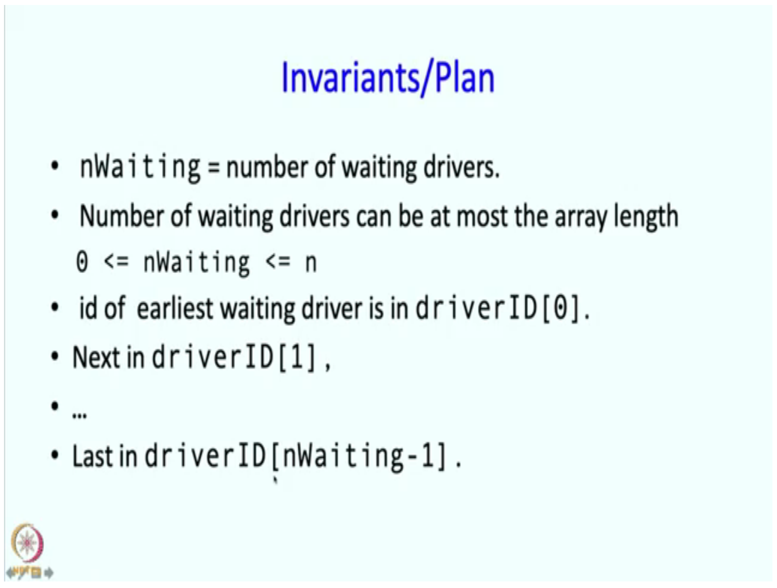
```
const int n=500; int driverID[n], nWaiting = 0;
while(true){
  char command; cin >> command;
  if(command == 'd'){
    // process driver arrival.
  }
  else if(command == 'c'){
    // process customer.
  }
  else if(command == 'x') break; // to terminate program
  else cout << "Illegal command.\n";
}
```

So, here is the outline of our program. So, we are going to define our arrays and there is the constant and then there is the variable nWaiting and we should at the beginning make nWaiting 0 because no driver has come in yet and our basic loop is going to be we are going to wait for the drivers or the customers to come. So, for this purpose the controller will type in a command. So, whatever the controller types in will be read into this variable called command.

So, this variable is going to be type char. So, our convention is going to be that if a driver has come in then the controller who is going to use our program will type in a d, d for driver. So, if the controller typed d then we need to somehow process that driver arrival. So, how to do that will decide in a minute. But if the command is a c, then we should process a customer arrival c for customer. So, our instructions to the controller are that if you see a driver come in and you want to tell the program that a driver has come in then you should type a d and then give the information needed corresponding to driver. If a customer comes in you should type a c and after that give the information corresponding to a customer. Otherwise if a command x is given, then we should stop. x starts for exit usually and in this case. And if you give any other command when the program will say this is a illegal command and it will not abort it will not stop but it will just go back and wait for another command. So, that is going to be our overall high level program.

So what am going to do over here is fill in all the details. So, how am i going to fill in all the details and this is a bit of a complicated program. So, let us really make a proper plan. Let us decide what exactly our variables are going to mean and then let us stick to those decisions. So, this is like what we have said long ago. So, before you write complicated things make a proper plan and right down invariants for our variables.

(Refer Slide Time: 08:21)



Invariants/Plan

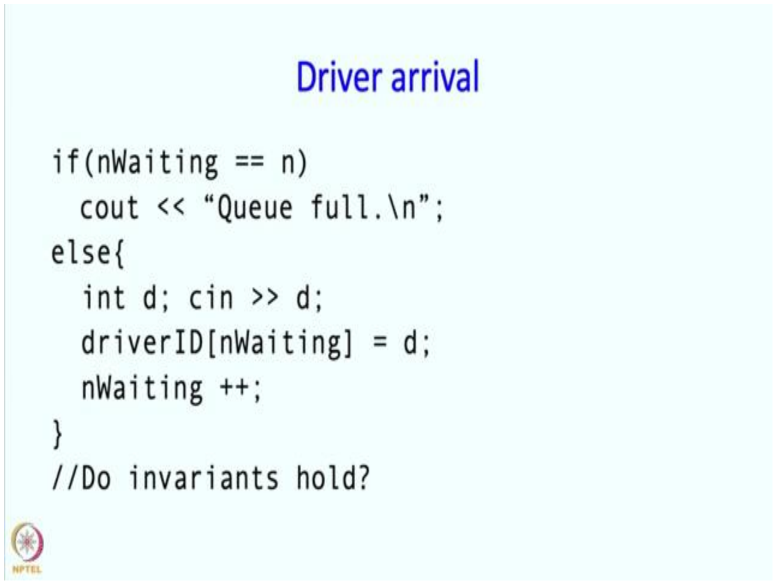
- nWaiting = number of waiting drivers.
- Number of waiting drivers can be at most the array length
  0 <= nWaiting <= n
- id of earliest waiting driver is in driverID[0].
- Next in driverID[1],
- …
- Last in driverID[nWaiting-1].

So, we have already talked about our variable nWaiting this is the number of waiting drivers. So, we want this variable to take a value in the range 0 through n. n is the largest number of drivers that we can have waiting. So, we should be careful that we never increase nWaiting beyond n. Then we should know where the ID of the earliest waiting driver is going to be.

So, let us make our plan and let us make a decision, let us say that the ID of the earliest waiting driver is always going to be in driverID[0]. Then the next driver is going to be in driverID[1] and so on. The last driver is going to be driverID[nWaiting-1]. Why this position well there are nWaiting drivers. So, starting here all the way till this you will have exactly nWaiting slots. So, if i want to insert a new driver who has just arrived where should i put the driver that driver should be put in driverID[nWaiting] and so on. The last driver is going to be driverID[nWaiting-1]. Why this position well there are nWaiting drivers. So, starting here all the way till this you will have exactly nWaiting slots. So, if i want to insert a new driver who has just arrived where should i put the driver. That driver should be put in driverID[nWaiting] and so on. The last driver is going to be driverID[nWaiting-1]. Why this position well there are nWaiting drivers. So, starting here all the way till this you will have exactly nWaiting slots.

So, if i want to insert a new driver who has just arrived where should i put the driver that driver should be put in driverID[nWaiting] and so on. The last driver is going to be driverID[nWaiting-1]. Why this position well there are nWaiting drivers. So, starting here all the way till this you will have exactly nWaiting slots. So, if i want to insert a new driver who has just arrived where should i put the driver that driver should be put in driverID[nWaiting]. Alright so what happens when driver arrival if nWaiting is n.

(Refer Slide Time: 09: 47)



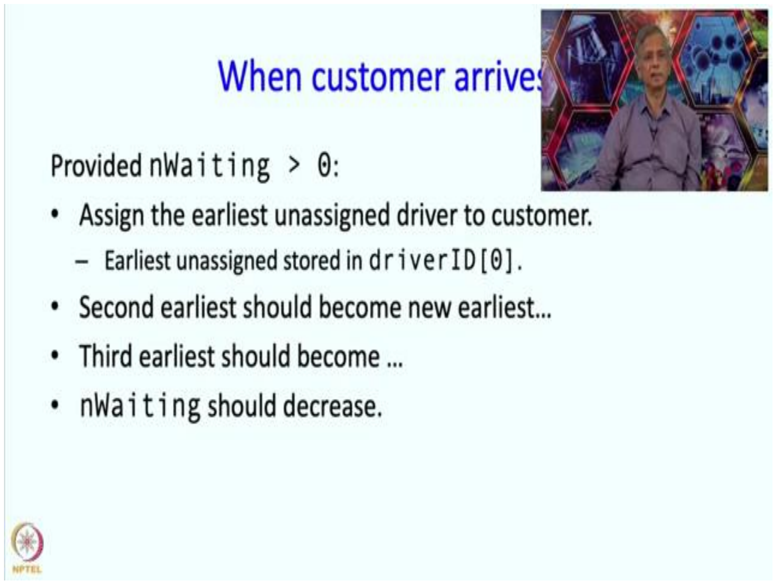
Driver arrival

```
if(nWaiting == n)
  cout << "Queue full.\n";
else{
  int d; cin >> d;
  driverID[nWaiting] = d;
  nWaiting ++;
}
//Do invariants hold?
```

And then i am going to say that is the queue is full. Otherwise we are going to get driver ID for this let us say we have located variable D and then as we have just said that this information goes into driverID[nWaiting]. So, this D is put over there. So, this is the ID of the last driver that have just arrived, but because the driver have just arrived what has happen well the number of waiting drivers has increased.

Therefore, we should increase nWaiting. That is it, that what we need to do if a driver has arrived. Of course, we have made some invariants, what were the invariants? We should make sure that nWaiting never goes beyond n. So, if you go through this program you will see that we incremented only if nWaiting is smaller than n and therefore it will never go. Similarly the other invariants you should also check they will also hold. What happens when a customer arrives?

(Refer Slide Time: 10: 55)



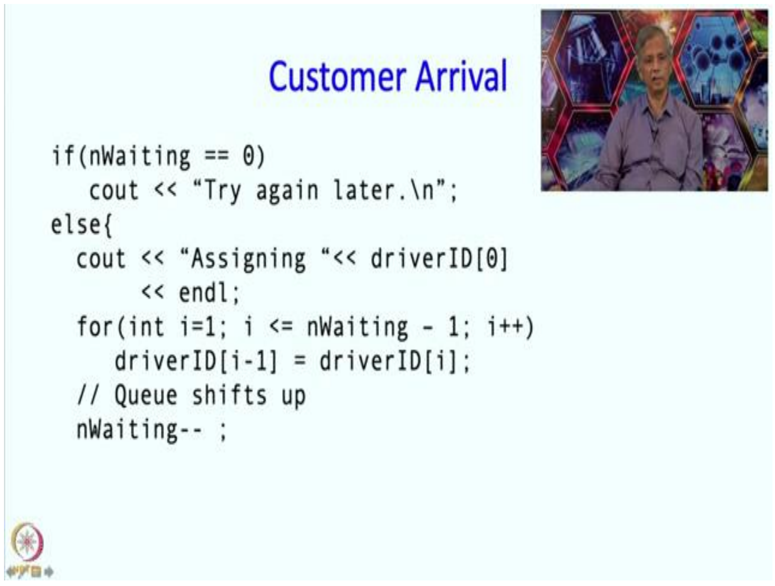
When customer arrive

Provided `nWaiting > 0`:

- Assign the earliest unassigned driver to customer.
  - Earliest unassigned stored in `driverID[0]`.
- Second earliest should become new earliest...
- Third earliest should become ...
- `nWaiting` should decrease.

Well what we need to do is something like this, we can process the customer only if nWaiting is bigger than 0. There had better be some taxi driver waiting if we want to do something with these customers. Otherwise we have to just tell the customers that try again but if nWaiting is bigger than 0 then we have to assign the earliest and unassigned the driver to the customers. The earliest unassigned driver is stored ID of that driver stored in driverID[0]. So, we can pick that up and give it. But now the second earliest should become the new earliest because the earliest has gone away. So, of all the drivers that are waiting the one that was the second earliest before this driver went away should now become the earliest and so on. So, we need to take care of that. And also nWaiting should decrease. So, these are the things that should happen according to the plan we have made.

(Refer Slide Time: 12: 07)

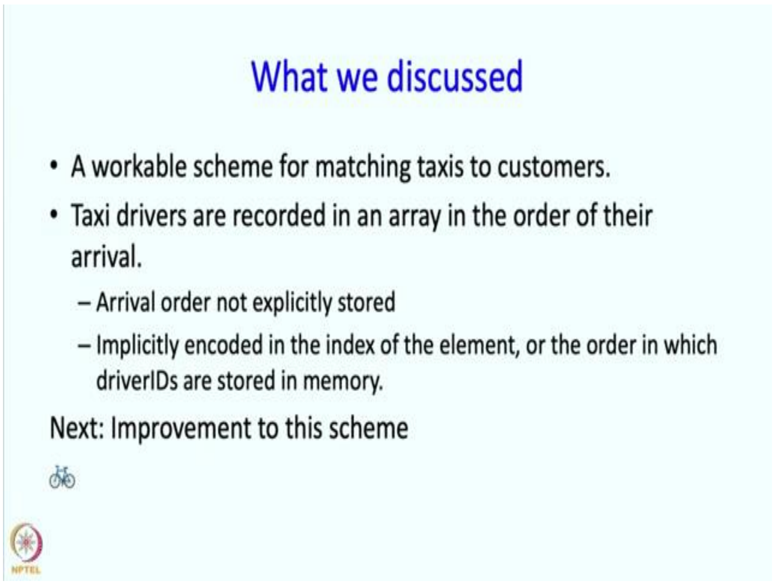


Customer Arrival

```
if(nWaiting == 0)
   cout << "Try again later.\n";
else{
  cout << "Assigning "<< driverID[0]
       << endl;
  for(int i=1; i <= nWaiting - 1; i++)
     driverID[i-1] = driverID[i];
  // Queue shifts up
  nWaiting-- ;
```

So, if nWaiting is 0 then we tell the customer please try again. Else, we are going to say I am going to assign the driver for this customer and we will just print the driver ID. The ID of the driver waiting in the 0 position. Will this ID be actually the ID of driver? Yes because nWaiting cannot be 0 nWaiting 0 case went away over here we know nWaiting has to be bigger than 0. So, this is a valid driver ID. And then we need to satisfy our other invariants which are that the waiting driver should be in position 0 through nWaiting-1. So, the position 0 driver has gone away, so, we should shift down all the waiting drivers 1 in this queue. So, to say that we are maintaining. So, this is what this loop accomplishes. So, the driver in position i at index i that ID is moved to position i minus1. And after that we decrement nWaiting. We write nWaiting=nWaiting-1 or nWaiting--. So, that is it

(Refer Slide Time: 13: 37)



What we discussed

- A workable scheme for matching taxis to customers.
- Taxi drivers are recorded in an array in the order of their arrival.
  - Arrival order not explicitly stored
  - Implicitly encoded in the index of the element, or the order in which driverIDs are stored in memory.

Next: Improvement to this scheme

So, what we have discussed. We have discussed a workable scheme for matching taxis to customers. So, taxi drivers are recorded in an Array in the order of their arrival and arrival order is not explicitly stored. So, we are not saying that this is zeroth the driver that arrived the first and the driver that arrived the second. We are not storing this numbers 1 and 2 just because those numbers are in a certain order in our array we know when they have or in the order in which they have arrived. In the next segment we are going to see some improvements to this scheme but we will take a quick break.