Computer Programming

Dr. Deepak B Phatak
Dr. Supratik Chakraborty
Department of Computer Science and Engineering
IIT Bombay

Session: Programming using Structures

Quick Recap of Relevant Topics

- Structures as collections of variables/arrays/other structures
- Pointers to structures
- Accessing members of structures
- Linked structures
- Dynamic allocation/de-allocation of structures

Overview of This Lecture

- Example C++ program using structures
  - Linked structures
  - Accessing members through “.” and “->”
  - Dynamic allocation/de-allocation of structures

Acknowledgment

- Some examples in this lecture are from

  An Introduction to Programming Through C++

  by Abhiram G. Ranade

  McGraw Hill Education 2014

- All such examples indicated in slides with the citation **AGRBook**

A Taxi Queuing System [Inspired by AGRBook]

- Taxis at a train station arrive and depart at different times
- We want to maintain a queue of taxis at the train station
- The queue implements a first-in-first-out policy for taxis
  - The taxi that came to the station earliest (among all waiting taxis) is at the “front” of the queue.
  - The taxi at the “front” of the queue is the next one to be dispatched.
  - The latest taxi to arrive at the station joins the queue at its “end”.
  - No taxi leaves the queue once it joins at the “end” until it reaches the “front”, and is dispatched.

A Taxi Queuing System: Structures Used

IIT Bombay

```
#include <iostream>
using namespace std;

struct Driver {string name; int id;};
struct LinkedTaxi {int id; Driver *drv; LinkedTaxi *next;};
struct Queue {LinkedTaxi *front, *end; int numTaxis;};
```

All structure definitions before and outside definitions of functions that use them

… Rest of program file …

A Taxi Queuing System: Structures Used

IIT Bombay

```
#include <iostream>
using namespace std;

struct Driver {string name; int id;};
struct LinkedTaxi {int id; Driver *drv; LinkedTaxi *next;};
struct Queue {LinkedTaxi *front, *end; int numTaxis;};
```

Note the use of "string" data type instead of "char" array. Helps simplify reading of strings with spaces using "cin"

… Rest of program file …

A Taxi Queuing System: Initializing The Queue


```
#include <iostream>
using namespace std;
… Structure definitions …
int main() {
   Queue q;
   q.front = NULL;  q.end = NULL; q.numTaxis = 0;
    … Rest of code …
}
```

A Taxi Queuing System: Main Loop


```
while (true) {
   cout << "Command: 'j' to join queue, 'd' to dispatch, 'x' to exit." << endl;
   char command;  cin >> command;
   switch(command) {
     case 'j':   … Code to add a newly arrived taxi at end of queue …
                 break;
     case 'd':  … Code to dispatch taxi at front of queue …
                 break;
     case 'x': cout << "Thank you" << endl; return 0;
     default: cout << "Invalid command." << endl;
   }
}
```

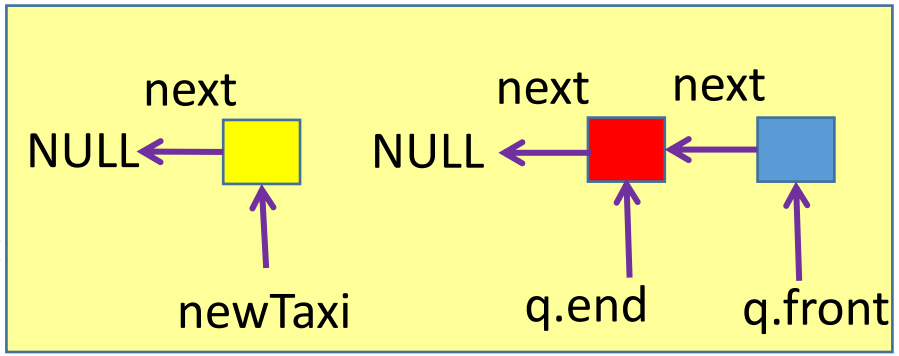
Adding a Taxi at “end” of Queue


```
Driver *newDrv;   newDrv = new Driver;
if (newDrv == NULL) {cout << “Memory allocation failure” << endl; return -1;}
cout << “Give name of driver: “;
cin >> newDrv->name;
cout << “Give id of driver: “;  cin >> newDrv->id;
LinkedTaxi *newTaxi;   newTaxi = new LinkedTaxi;
if (newTaxi == NULL) {cout << “Memory allocation failure”<< endl; return -1;}
newTaxi->drv = newDrv;  newTaxi->next = NULL;
cout << “Give id of taxi: “;  cin >> newTaxi->id;
if (q.end == NULL) { // Taxi queue empty
  q.front = newTaxi; q.end = newTaxi; q.numTaxis = 1;
}
else {(q.end)->next = newTaxi;  q.end = newTaxi; q.numTaxis++; }
```

Adding a Taxi at "end" of Queue

IIT Bombay

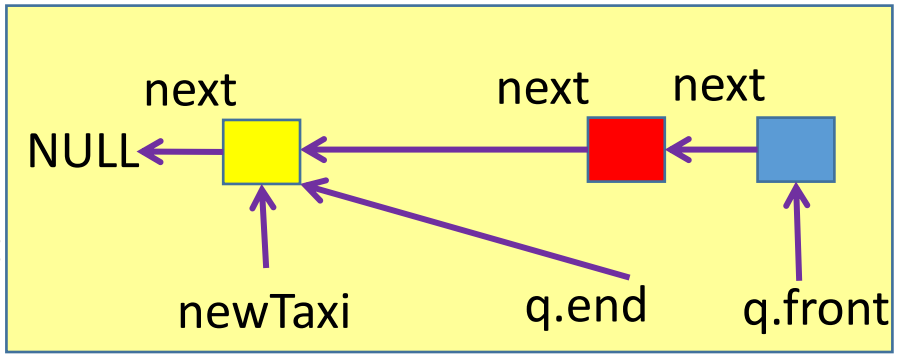
```
Driver *newDrv;   newDrv = new Driver;
if (newDrv == NULL) {cout << "Memory allocation failure" << endl; return -1;}
cout << "Give name of driver: ";
cin >> newDrv->name;
cout << "Give id of driver: ";  cin >> newDrv->id;
LinkedTaxi *newTaxi;   newTaxi = new LinkedTaxi;
if (newTaxi == NULL) {cout << "Memory allocation failure"<< endl; return -1;}
newTaxi->drv = newDrv;  newTaxi->next = NULL;
cout << "Give id of taxi: ";  cin >> newTaxi->id;
if (q.end == NULL) { // Taxi queue empty
   q.front = newTaxi; q.end = newTaxi; q.numTaxis = 1;
}
else {(q.end)->next = newTaxi;  q.end = newTaxi; q.numTaxis++; }
```



Adding a Taxi at "end" of Queue

IIT Bombay

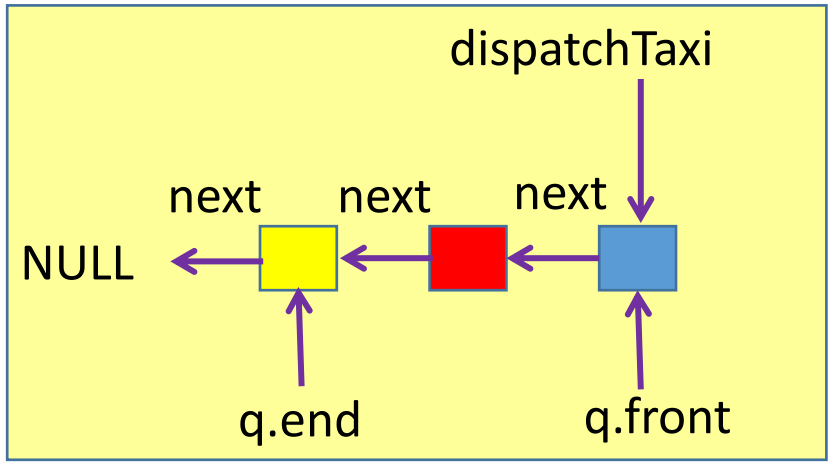
```
Driver *newDrv;   newDrv = new Driver;
if (newDrv == NULL) {cout << "Memory allocation failure" << endl; return -1;}
cout << "Give name of driver: ";
cin >> newDrv->name;
cout << "Give id of driver: ";  cin >> newDrv->id;
LinkedTaxi *newTaxi;   newTaxi = new LinkedTaxi;
if (newTaxi == NULL) {cout << "Memory allocation failure"<< endl; return -1;}
newTaxi->drv = newDrv;  newTaxi->next = NULL;
cout << "Give id of taxi: ";  cin >> newTaxi->id;
if (q.end == NULL) { // Taxi queue empty
   q.front = newTaxi; q.end = newTaxi; q.numTaxis = 1;
}
else {(q.end)->next = newTaxi;  q.end = newTaxi; q.numTaxis++; }
```



Dispatching a Taxi from “front” of Queue

IIT Bombay

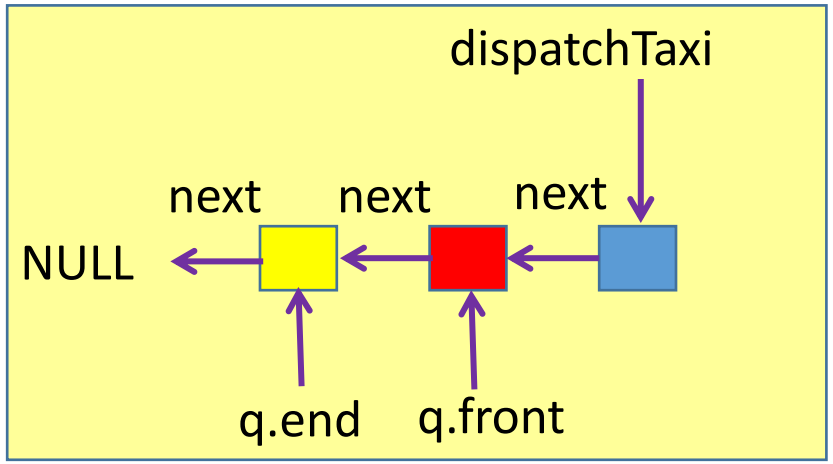
```
if (q.front == NULL) {cout << “Sorry! No taxis in queue at present!” << endl;}
else {
  LinkedTaxi *dispatchTaxi; dispatchTaxi = q.front;
  if (q.front == q.end) { // Only one taxi in queue
     q.front = NULL; q.end = NULL; q.numTaxis = 0;
  }
   else {q.front = (q.front)->next;  q.numTaxis--;}
  if (dispatchTaxi != NULL) {
    cout << “Dispatching taxi with id “ << dispatchTaxi->id << endl;
    if (dispatchTaxi->drv != NULL) {delete dispatchTaxi->drv;}
   delete dispatchTaxi;
  }
}
```



Dispatching a Taxi from “front” of Queue

IIT Bombay

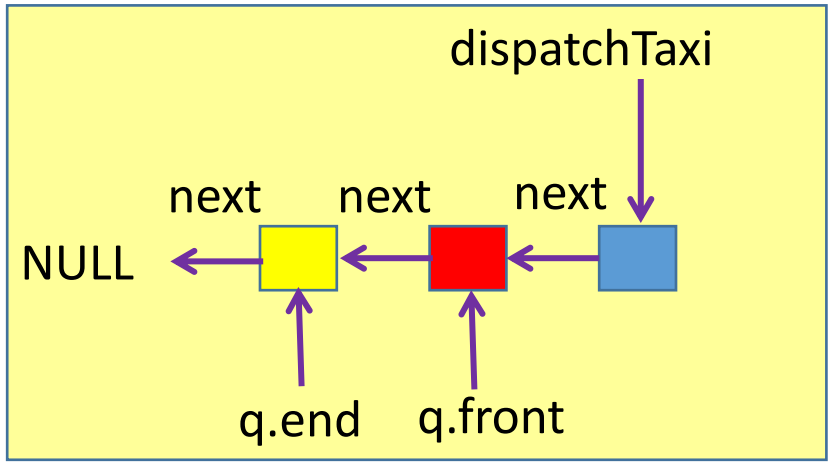
```
if (q.front == NULL) {cout << “Sorry! No taxis in queue at present!” << endl;}
else {
  LinkedTaxi *dispatchTaxi; dispatchTaxi = q.front;
  if (q.front == q.end) { // Only one taxi in queue
     q.front = NULL; q.end = NULL; q.numTaxis = 0;
  }
   else {q.front = (q.front)->next;  q.numTaxis--;}
  if (dispatchTaxi != NULL) {
    cout << “Dispatching taxi with id “ << dispatchTaxi->id << endl;
    if (dispatchTaxi->drv != NULL) {delete dispatchTaxi->drv;}
   delete dispatchTaxi;
  }
}
```



Dispatching a Taxi from “front” of Queue

IIT Bombay

```
if (q.front == NULL) {cout << “Sorry! No taxis in queue at present!” << endl;}
else {
  LinkedTaxi *dispatchTaxi; dispatchTaxi = q.front;
  if (q.front == q.end) { // Only one taxi in queue
     q.front = NULL; q.end = NULL; q.numTaxis = 0;
  }
   else {q.front = (q.front)->next;  q.numTaxis--;}
  if (dispatchTaxi != NULL) {
    cout << “Dispatching taxi with id “ << dispatchTaxi->id << endl;
    if (dispatchTaxi->drv != NULL) {delete dispatchTaxi->drv;}
    delete dispatchTaxi;
  }
}
```



Summary

- A taxi dispatch system implementing a first-in-first-out order (queue)
- Use of
  - Linked structures
  - Dynamic allocation/de-allocation of structures
  - Accessing members through “.” and “->” operators
- Size of queue not pre-determined
  - Amount of memory in system dictates this