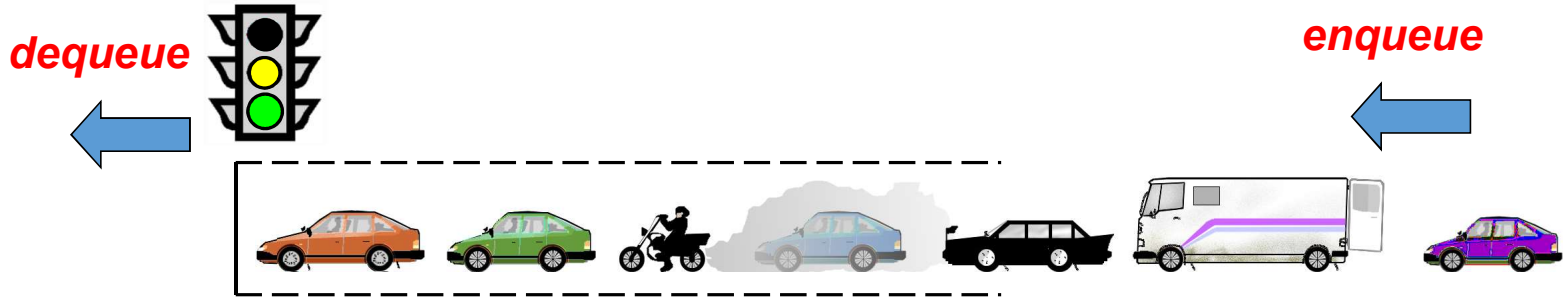
Queue

What is a Queue?

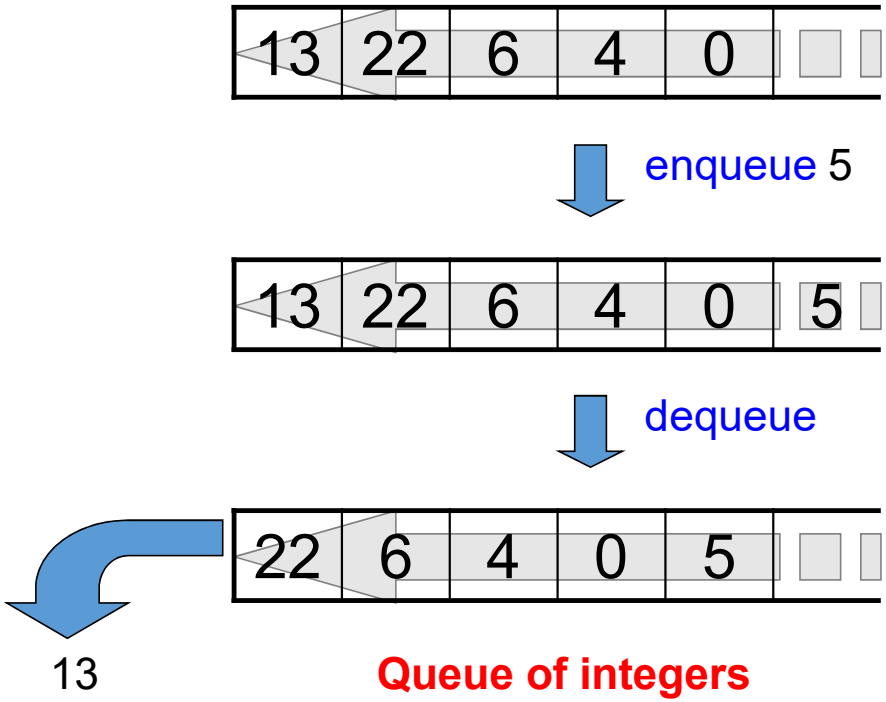
- A ***queue*** is a sequence of objects which is "first-come-first-served."



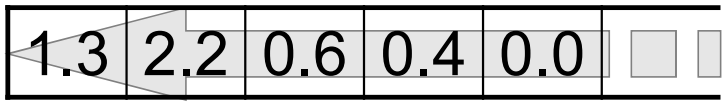
- Operations on a queue:
  - ***Enqueue***: joining a queue at the *end*.
  - ***Dequeue***: removing an object from the *front*.

Queue

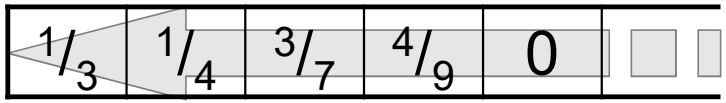
- As a data structure, a queue is used to store values.



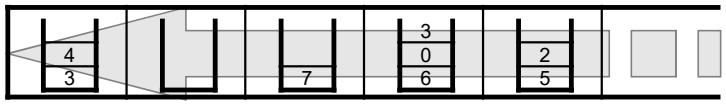
Queue



Queue of real numbers



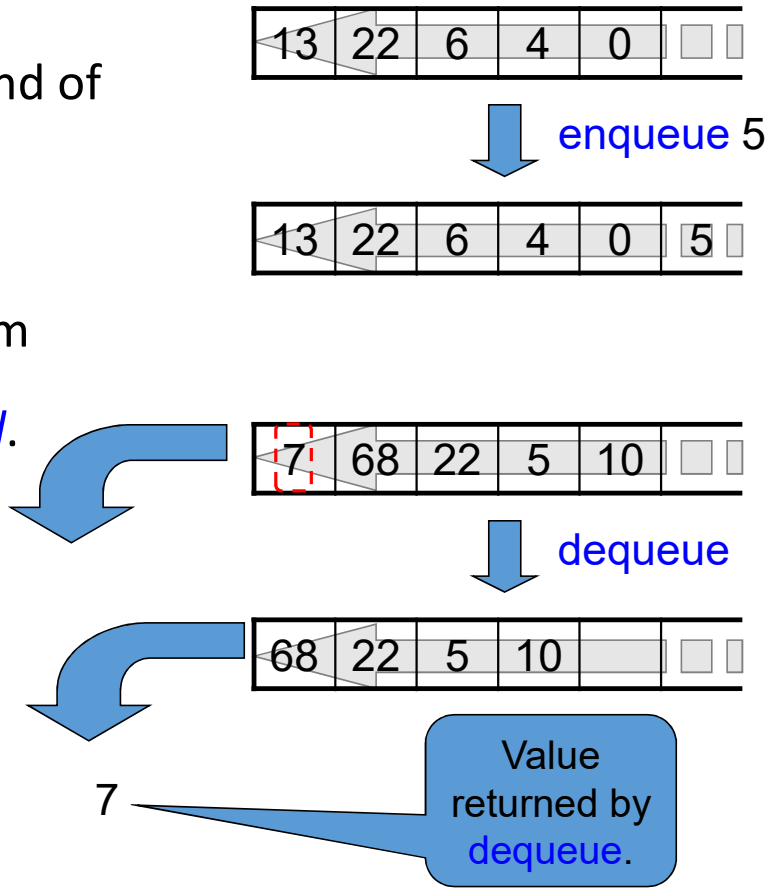
Queue of rational numbers



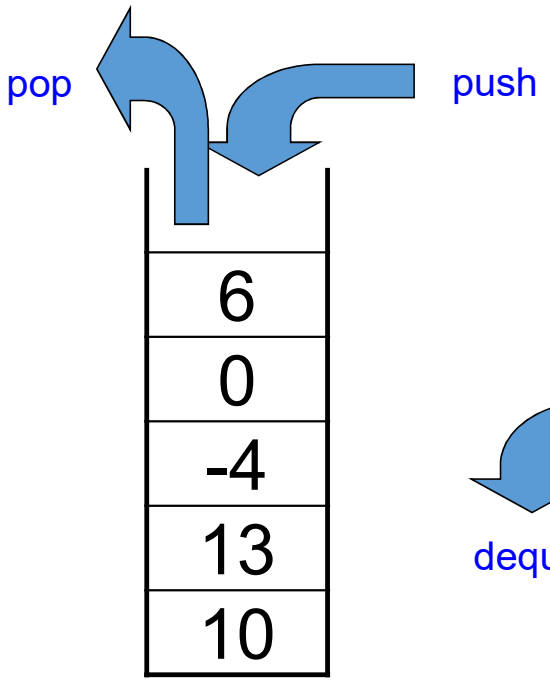
Queue of stack of integers

Queue Operations: Enqueue and Dequeue

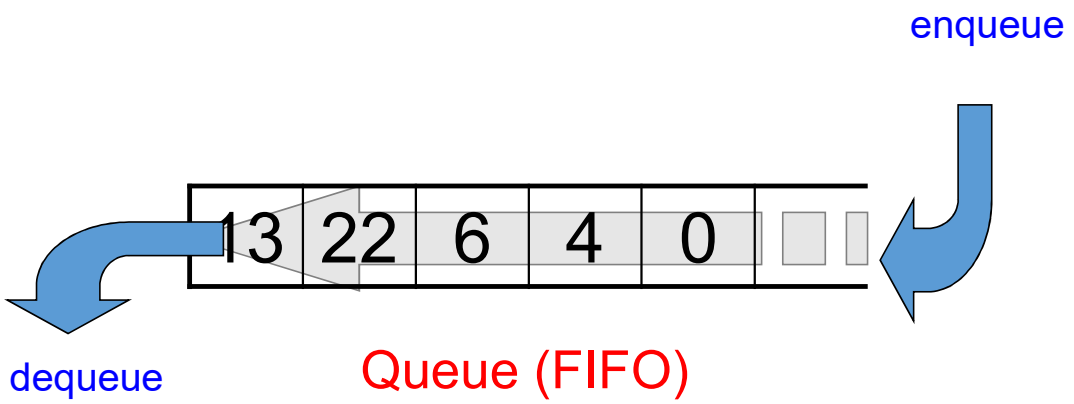
- Enqueue
  - Puts a value at the end of a queue.
- Dequeue
  - Removes a value from the front of a queue.
  - The value is *returned*.
- A queue is a ***First-In-First-Out*** (FIFO) data structure.



Stack vs Queue

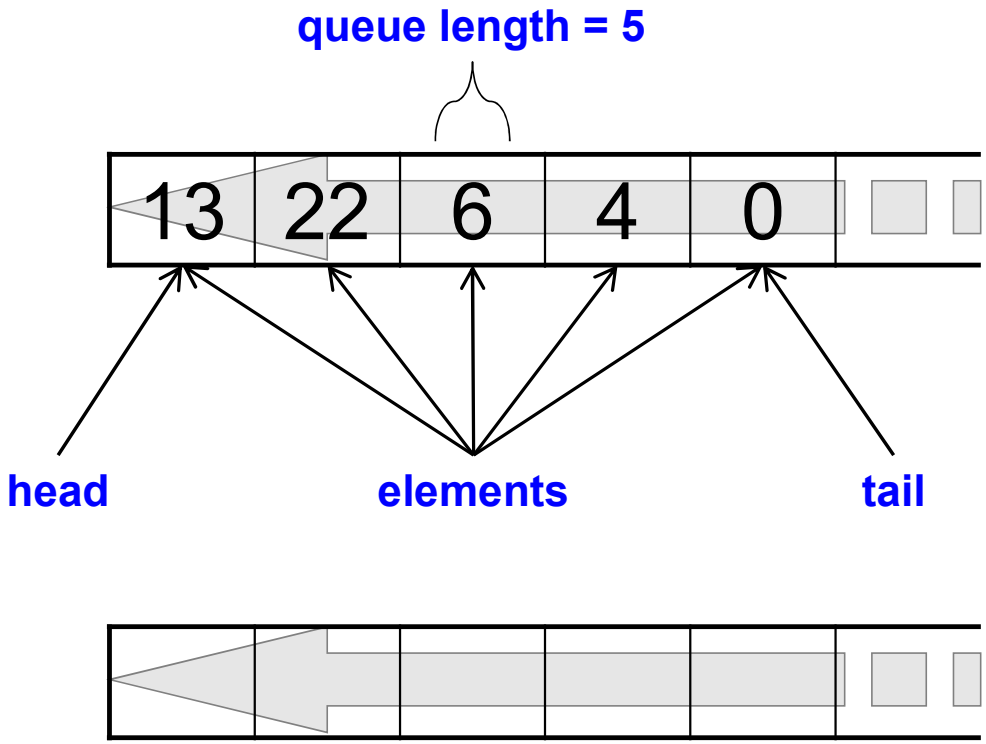


Stack (LIFO)



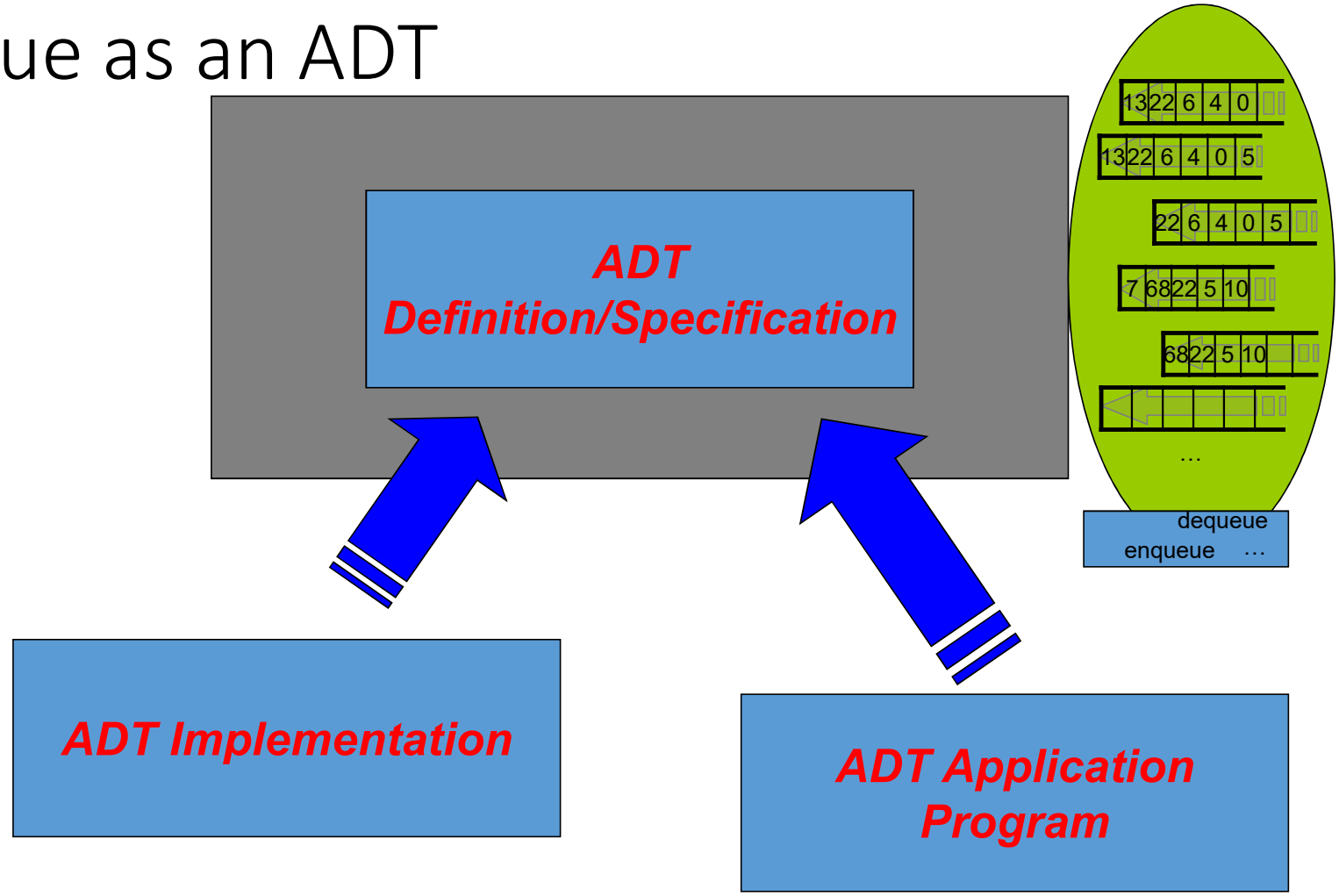
Queue (FIFO)

Queue Terminologies



empty queue (length = 0)

Queue as an ADT



Defining a Queue ADT: queue.h

- Everything looks so familiar…

```
typedef struct queueCDT *queueADT;

typedef int queueElementT;

queueADT EmptyQueue();
void Enqueue(queueADT queue, queueElementT element);
queueElementT Dequeue(queueADT queue);
int QueueLength(queueADT queue);
int QueueIsEmpty(queueADT queue);
```

Defining a Queue ADT

```
queueADT EmptyQueue();
```

- Creates and returns a new empty queue.

```
void Enqueue(queueADT queue,
             queueElementT element);
```

- Adds the element **`element`** to the *tail* of the queue **`queue`**. Nothing is returned.

Defining a Queue ADT

```
queueElementT Dequeue(queueADT queue);
```

- Removes an element from the *head* of `queue` and returns the element.

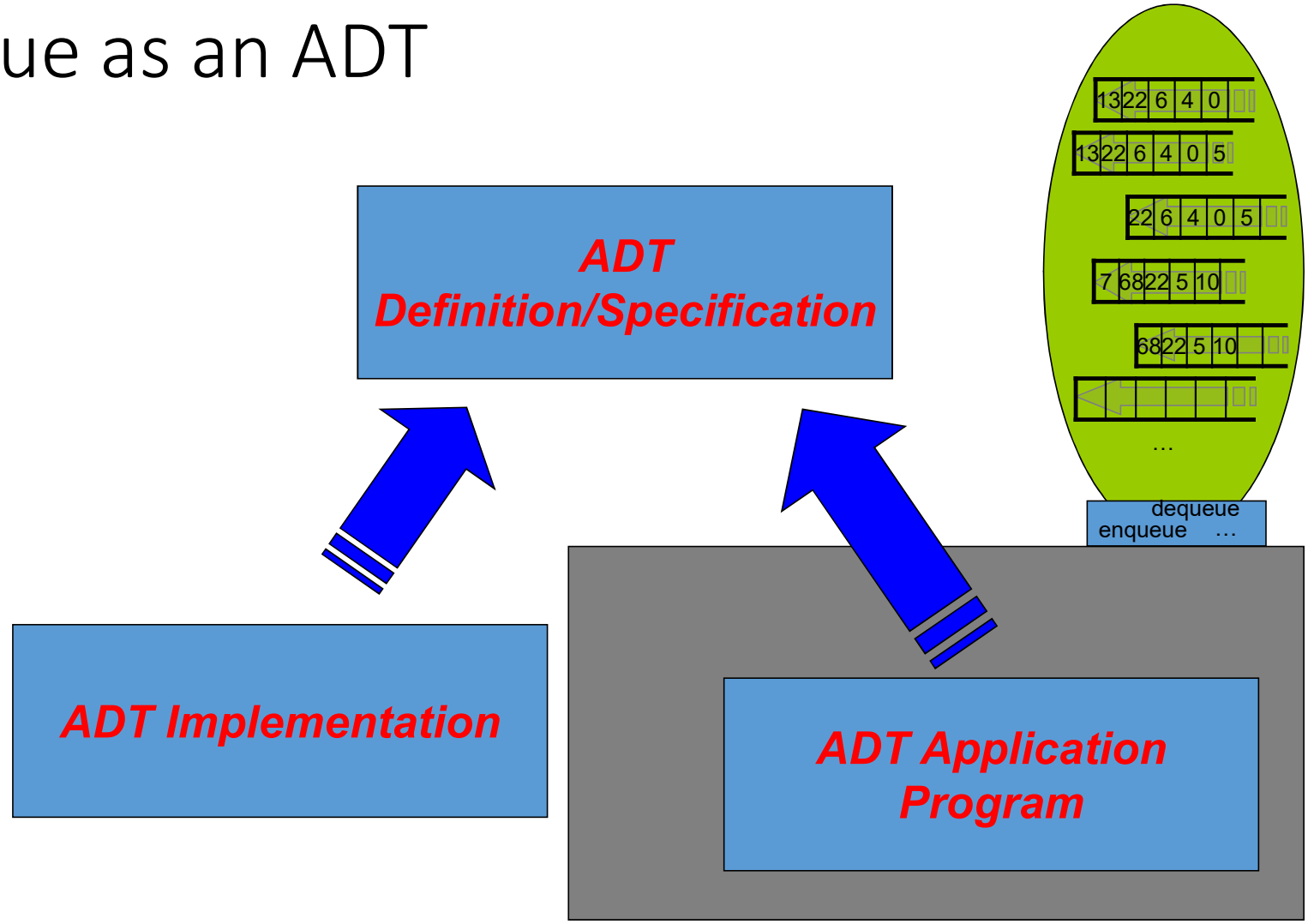
```
int QueueLength(queueADT queue);
```

- Returns the length of `queue`.

```
int QueueIsEmpty(queueADT queue);
```

- Returns 1 if `queue` is empty; 0 otherwise.

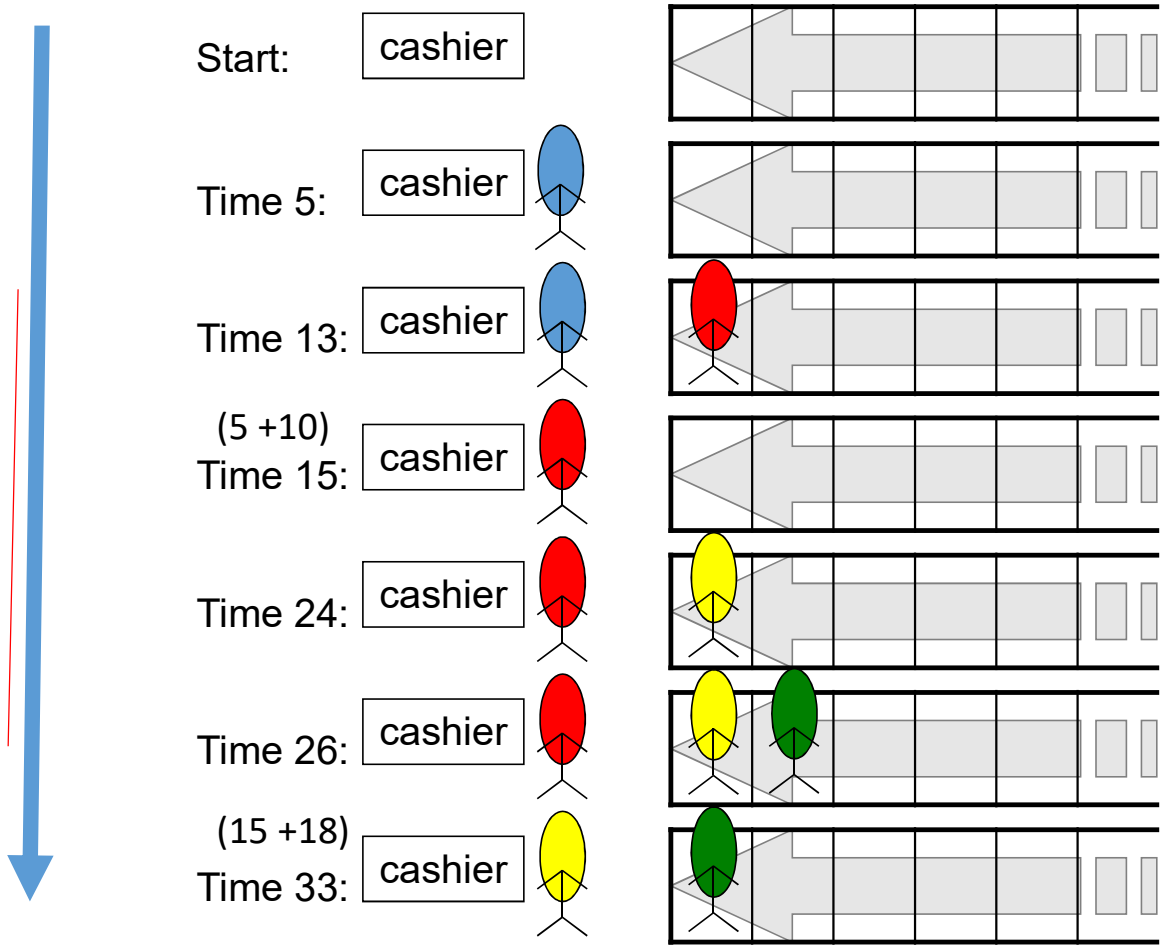
Queue as an ADT



A Queue *Application*: Supermarket Cashier Simulation

- In a supermarket
  - There is *one* cashier.
  - Customers go to the cashier for check-out after shopping.
  - If a cashier is occupied, a customer has to line up.
  - The cashier serves a customer in *10 to 50* seconds.

Supermarket Cashier Simulation



Customer	Arr.Time	Sv.Time
	5	10s
	13	18s
	24	22s
	26	33s

(33 +22)

Time 55: yellow leaves, serve green.

Time 88: green leaves.

(55 +33)

Supermarket Cashier Simulation

supermarket.c

```
#include "queue.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define ARRPROB 0.05    // customer arriving probability
#define SERVMIN 10      // min serving time
#define SERVMAX 50      // max serving time
#define SIMTIME 1000    // simulation time length

void simulateCustomerArrival(queueADT queue, int time);
void simulateCustomerLeaving(int time, int cashier);
int simulateCustomerServing(queueADT queue, int time,
                            int cashier);
```

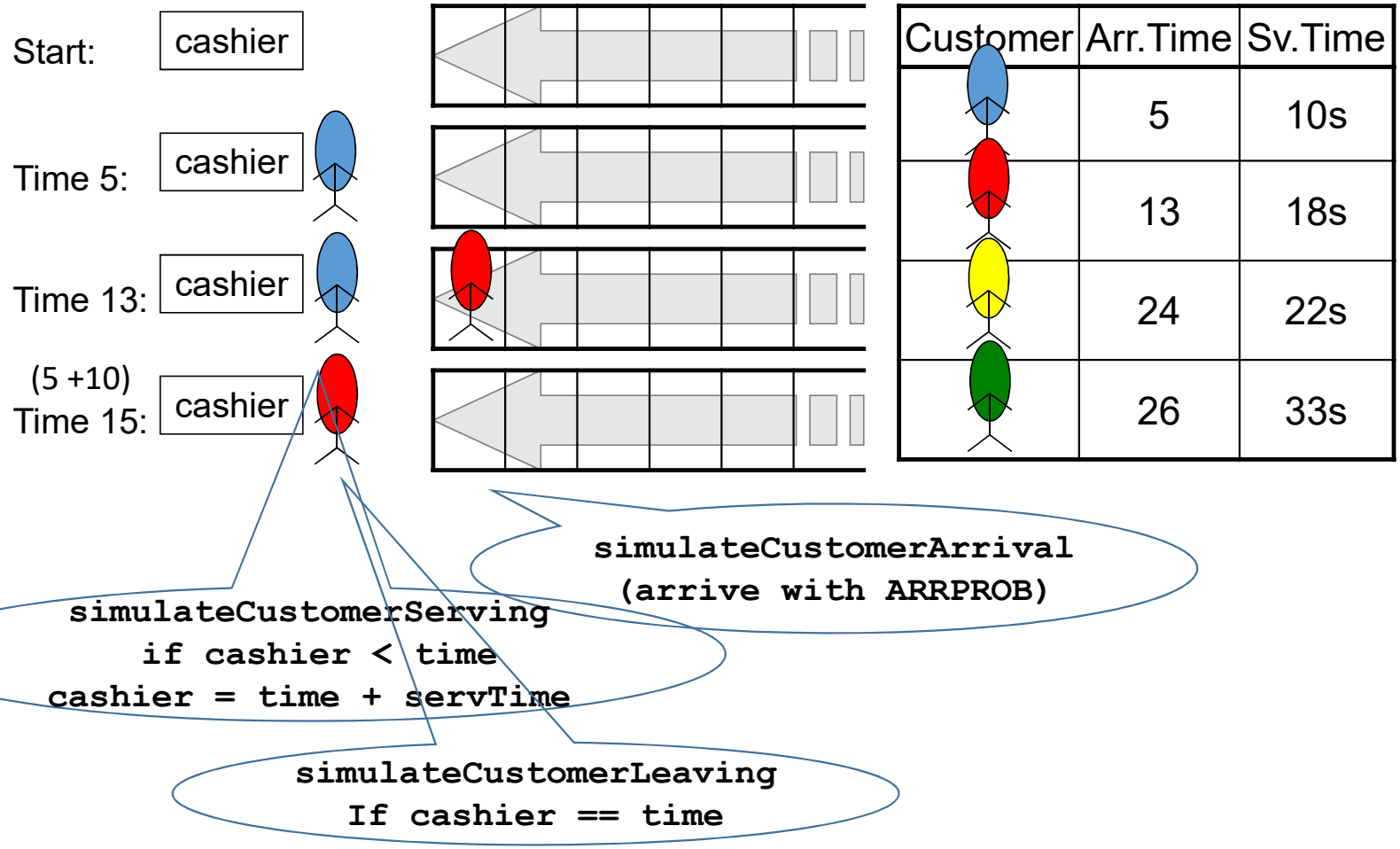
Supermarket Cashier Simulation

supermarket.c (continue)

```
int main() {
   int i, cashier = -1;
   queueADT customerQueue;

   srand(time(NULL));
   customerQueue = EmptyQueue();
   printf("Time\tEvent\t\t\tSv.Time\tQ.Length\n");
   printf("=========================");
   printf("=========================\n");
   for (i = 0; i < SIMTIME; i++) {
      simulateCustomerArrival(customerQueue, i);
      cashier = simulateCustomerServing(customerQueue,
                                        i, cashier);
      simulateCustomerLeaving(i, cashier);
   }
   return 0;
}
```

Stores the time *when* the cashier will be ready again.

Supermarket Cashier Simulation



Supermarket Cashier Simulation

supermarket.c (continue)

```
void simulateCustomerArrival(queueADT queue,
                             int time) {
   int servTime;

   if ((double)rand() / RAND_MAX <= ARRPROB) {
      servTime = rand() % (SERVMAX - SERVMIN + 1)
                 + SERVMIN;
      Enqueue(queue, servTime);
      printf("%d\tCustomer arrives\t%d\t%d\n", time,
             servTime, QueueLength(queue));
   }
}
```

Returns 1 with ARRPROB probability.

Generates a random number in [SERVMIN, SERVMAX].

Supermarket Cashier Simulation

supermarket.c (continue)

```
int simulateCustomerServing(queueADT queue, int time,
                            int cashier) {
   int servTime;

   if (cashier <= time && !QueueIsEmpty(queue)) {
      servTime = Dequeue(queue);
      printf("%d\tStart serving customer\t%d\t%d\n",
             time, servTime, QueueLength(queue));
      cashier = time + servTime;
   }
   return cashier;
}
```

Cashier is available.

Queue is not empty

Cashier remains occupied until time + servTime.

Supermarket Cashier Simulation

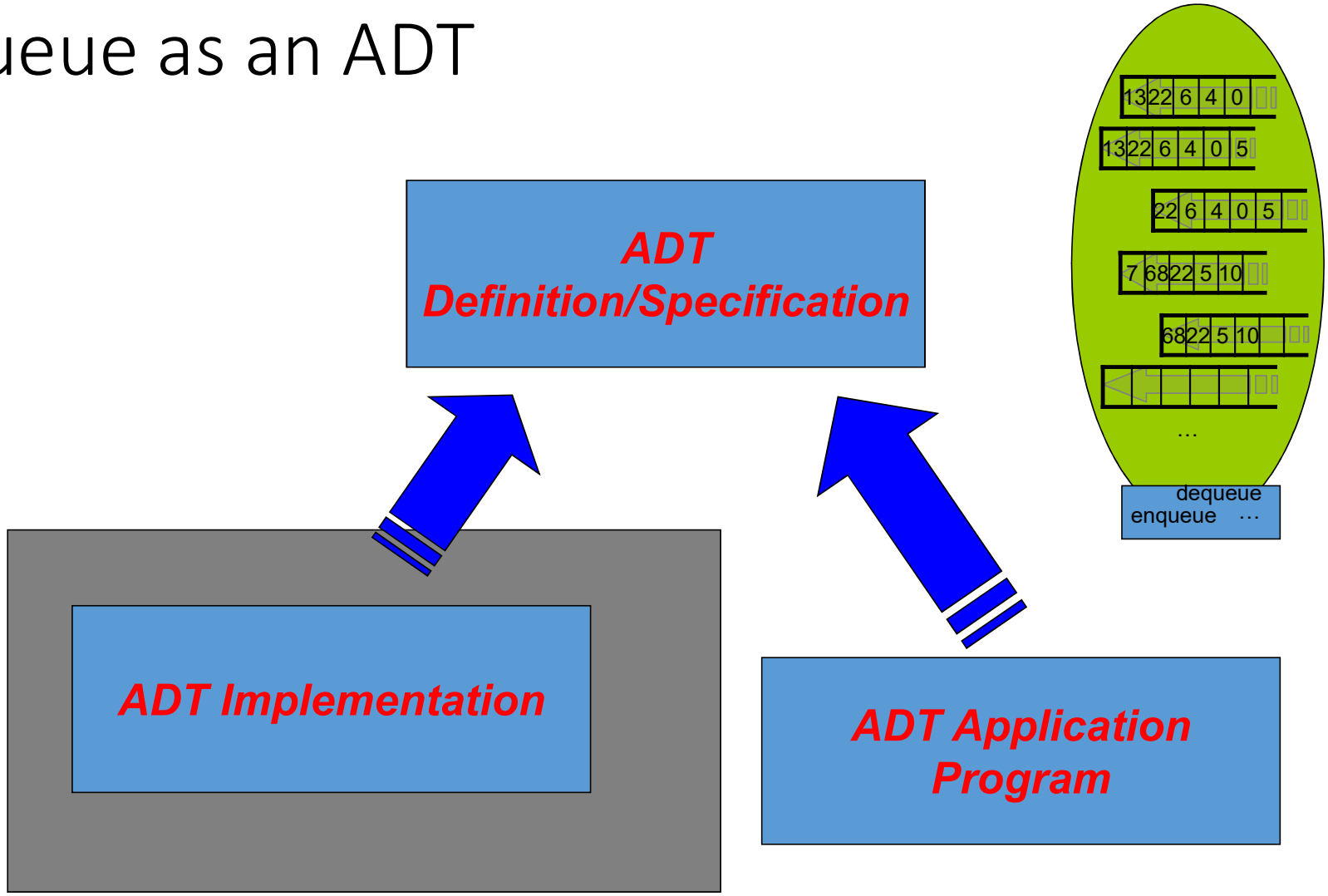
supermarket.c (continue)

```
void simulateCustomerLeaving(int time, int cashier) {
   if (cashier == time)
      printf("%d\tCustomer leaves\n", time);
}
```

Customer is time to leave.

Time	Event	Sv.Time	Q.Length
25	Customer arrives	28	1
25	Start serving customer	28	0
33	Customer arrives	40	1
53	Customer leaves		
53	Start serving customer	40	0
55	Customer arrives	34	1
62	Customer arrives	13	2
82	Customer arrives	36	3
93	Customer leaves		
93	Start serving customer	34	2
97	Customer arrives	24	3
101	Customer arrives	29	4
110	Customer arrives	33	5
114	Customer arrives	19	6
115	Customer arrives	10	7
127	Customer leaves		
127	Start serving customer	13	6
140	Customer leaves		
140	Start serving customer	36	5
.....			

Queue as an ADT



Implementing the Queue ADT

- We can also use arrays (either *fix-sized* or *dynamic*) to give implementation Versions *1.0* and *2.0*.

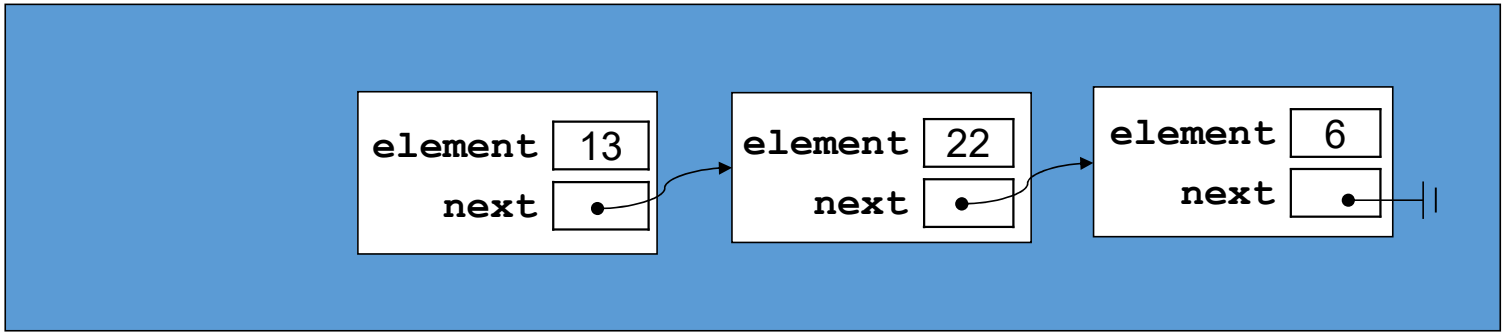
```
struct queueCDT {
   queueElementT elements[100];
   int head;
   int tail;
}
```

```
struct queueCDT {
   queueElementT *elements;
   int head;
   int tail;
   int size;
}
```

- Exercise
  - Complete the implementation of Versions 1.0 and 2.0.
- In Version ***3***.0, we introduce the use of ***linked list*** to implement the queue ADT.

A simple Link List

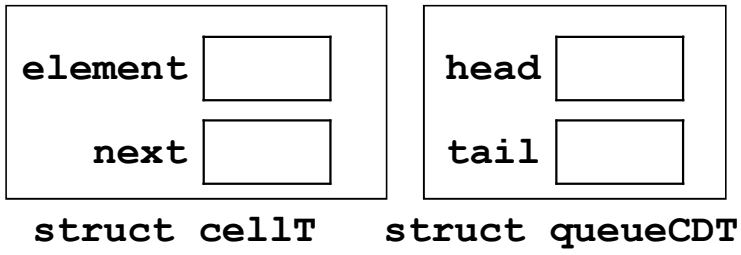
- A simple link list consists of a series of nodes
- Each node contains the element and a link to the next node (its successor)
- The last node links to the nullptr.



Queue *Implementation* (Ver 3.0)

- In Version ***3***.0, we introduce the use of ***linked list*** to implement the queue ADT.

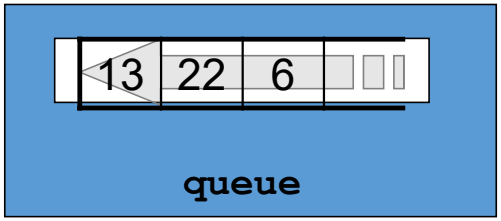
```
typedef struct cellT {
   queueElementT element;
   struct cellT *next;
} cellT;

struct queueCDT {
   cellT *head;
   cellT *tail;
};
```

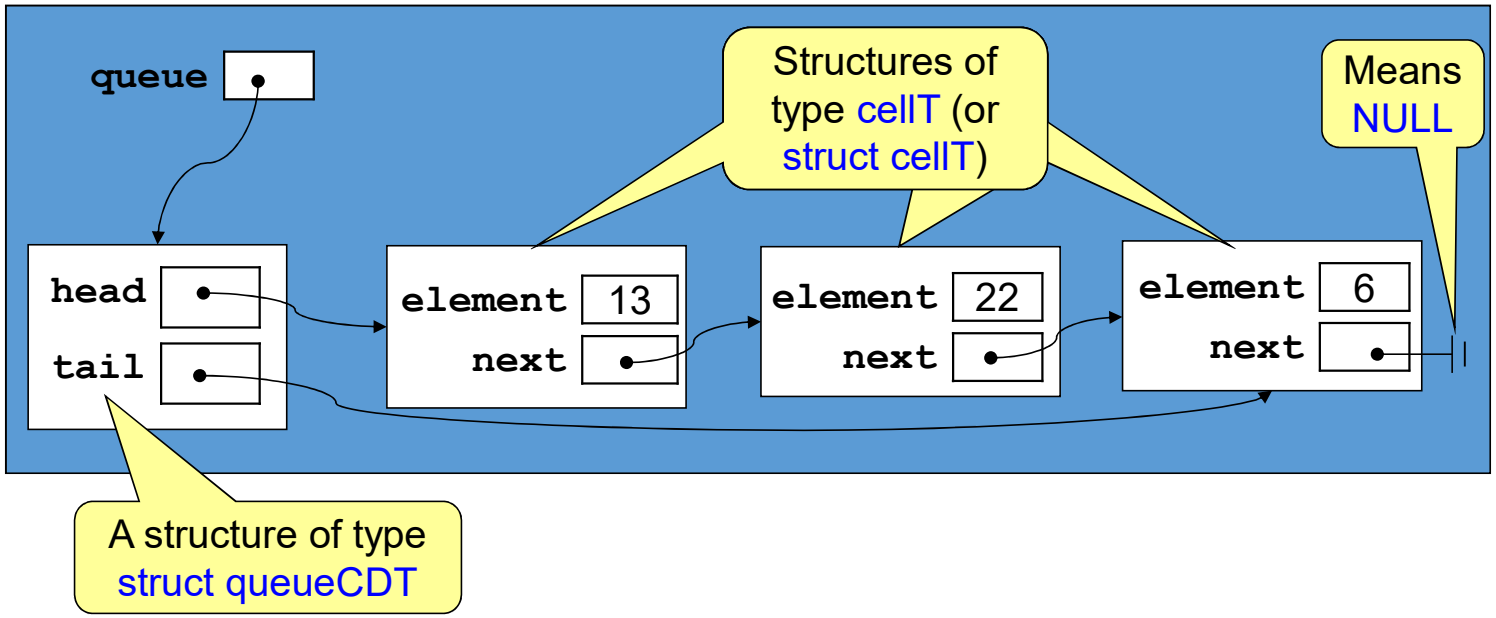


- A queue consists of *one* struct queueCDT, but *possibly many* struct cellTs.

Queue *Implementation* (Ver 3.0)

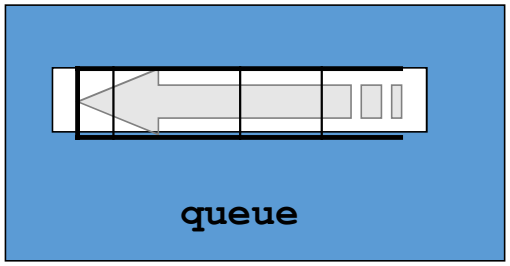
- A queue consists of *one* struct queueCDT, but *possibly many* struct cellTs.



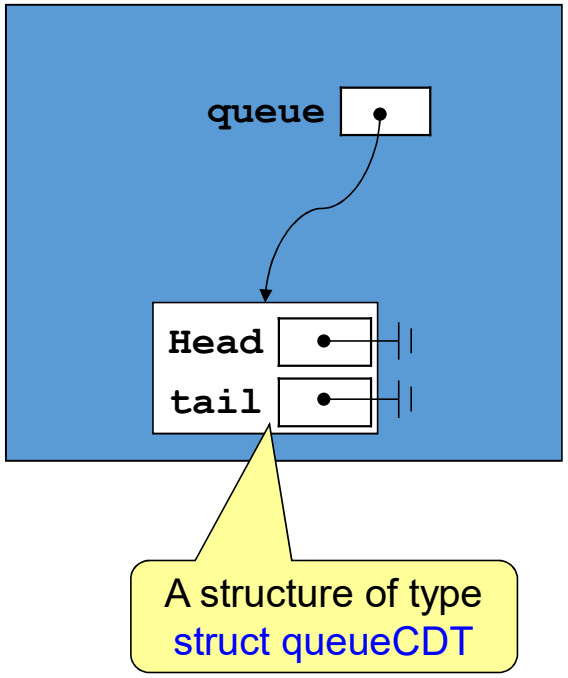


Queue *Implementation* (Ver 3.0)

- An *empty* queue consists of *one* struct queueCDT, but *no* struct cellTs.

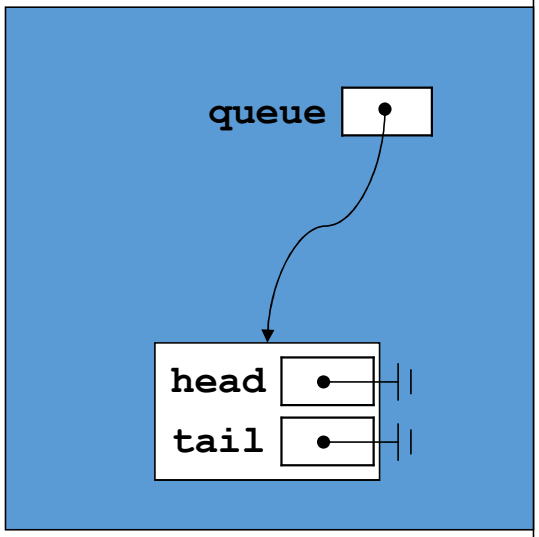


(empty queue)



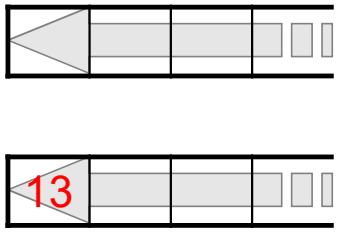
queue.c

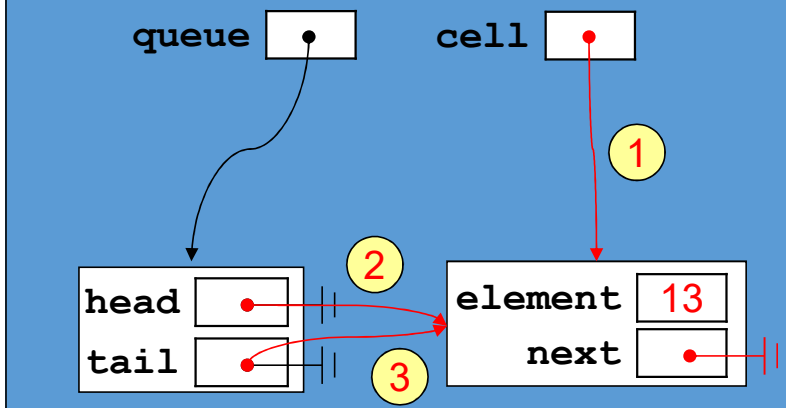
```
#include "queue.h"
#include <stdlib.h>

typedef struct cellT {
   queueElementT element;
   struct cellT *next;
} cellT;

struct queueCDT {
   cellT *head;
   cellT *tail;
}

queueADT EmptyQueue() {
   queueADT queue;
   queue = (queueADT)malloc(sizeof(struct queueCDT));
   queue->head = NULL;
   queue->tail = NULL;
   return queue;
}
```



queue.c (continue)

```
void Enqueue(queueADT queue, queueElementT element) {
   cellT *cell;
   cell = (cellT *)malloc(sizeof(cellT));   1
   cell->element = element;
   cell->next = NULL;
   if (QueueIsEmpty(queue))
      queue->head = cell;   2
   else
      queue->tail->next = cell;
   queue->tail = cell;   3
}
```

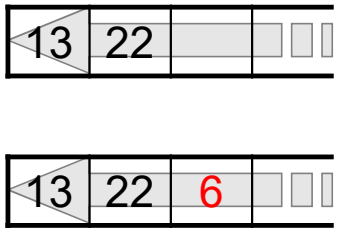


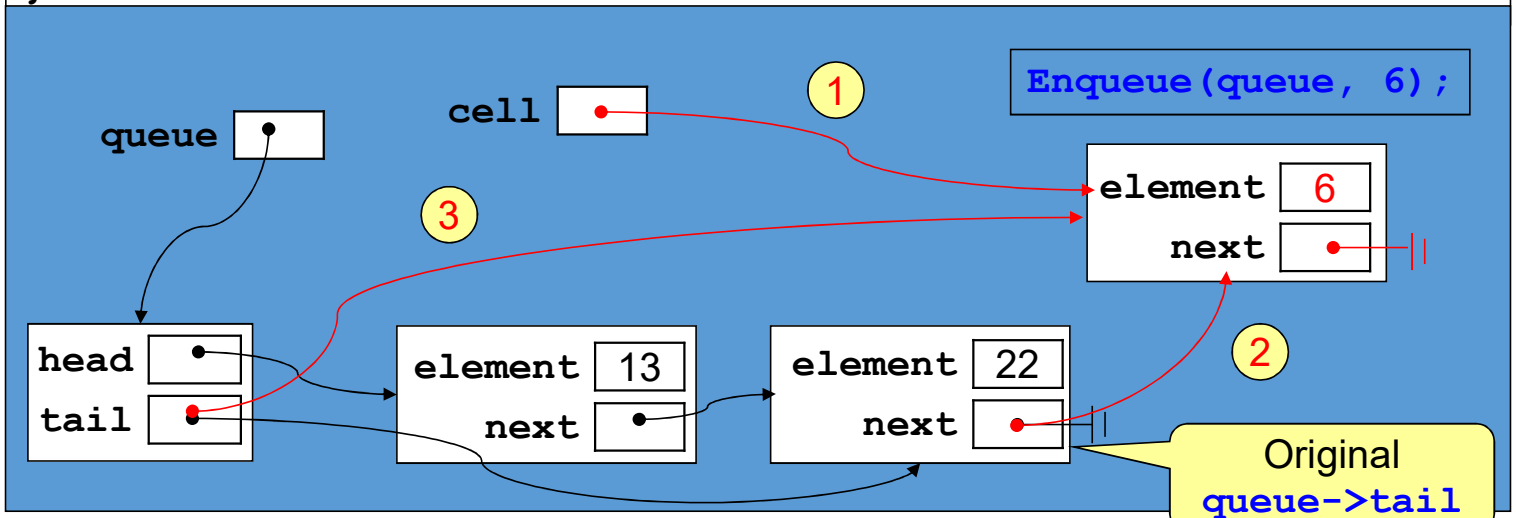



```
Enqueue(queue, 13);
```

queue.c (continue)

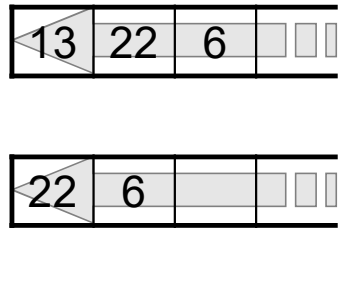
```
void Enqueue(queueADT queue, queueElementT element) {
   cellT *cell;
   cell = (cellT *)malloc(sizeof(cellT));   1
   cell->element = element;
   cell->next = NULL;
   if (QueueIsEmpty(queue))
      queue->head = cell;
   else
      queue->tail->next = cell;   2
   queue->tail = cell;   3
}
```

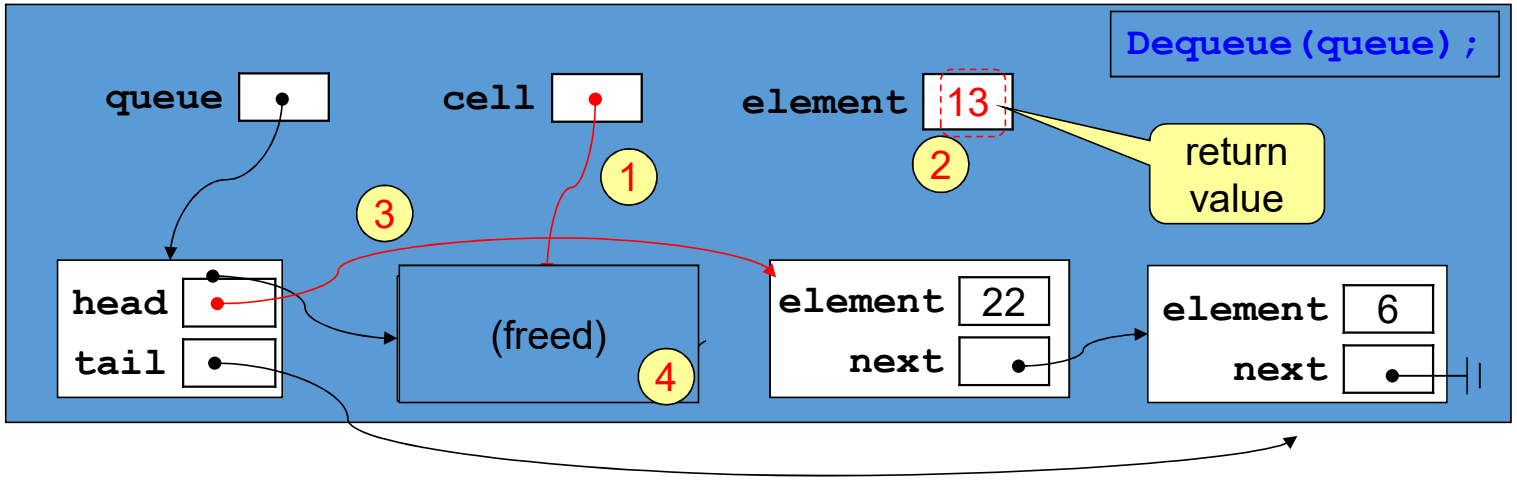




queue.c (continue)

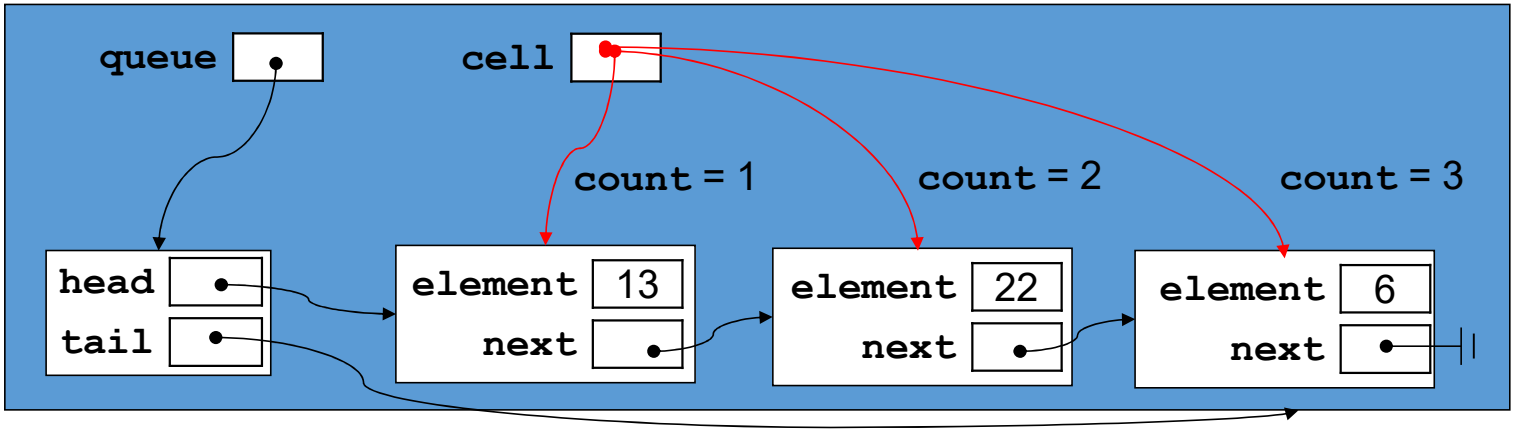
```
queueElementT Dequeue(queueADT queue) {
   queueElementT element;
   cellT *cell;
   if (QueueIsEmpty(queue))
      exit(0);
   cell = queue->head;
   element = cell->element;
   queue->head = cell->next;
   free(cell);
   return element;
}
```





queue.c (continue)

```
int QueueLength(queueADT queue) {
   cellT *cell;
   int count = 0;
   cell = queue->head;
   while (cell != NULL) {
      count++;
      cell = cell->next;
   }
   return count;
}
```



queue.c (continue)

```
int QueueIsEmpty(queueADT queue) {
   return (queue->head == NULL);
   // or: return (QueueLength(queue) == 0);
}
```

Done!

- Possible improvement (Version 3.***1***)
  - Add a member to struct queueCDT to store the number of elements in a queue (i.e., the queue length)

```
struct queueCDT {
   cellT *head;
   cellT *tail;
   int count;
};
```

The Complete Program