DYNAMIC DATA STRUCTURES II

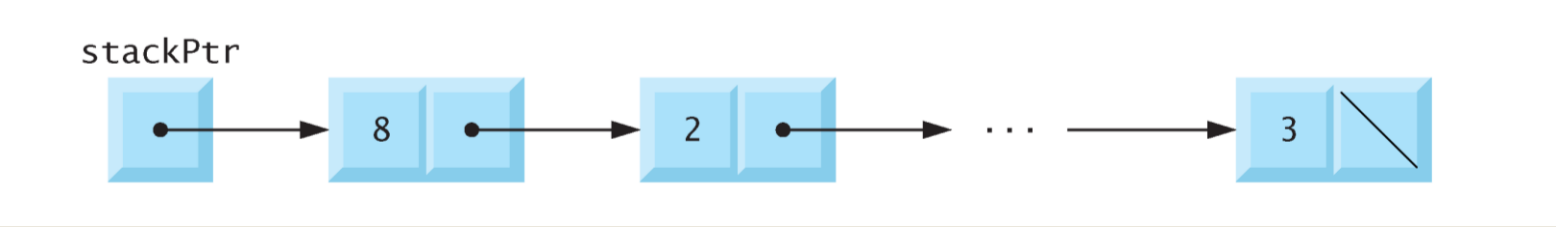
CEN116 Algorithms and Programming II

Stacks

- A stack can be implemented as a constrained version of a linked list.
- New nodes can be added to a stack and removed from a stack only at the top.
- For this reason, a stack is referred to as a last-in, first-out (LIFO) data structure.
- A stack is referenced via a pointer to the top element of the stack.
- The link member in the last node of the stack is set to NULL to indicate the bottom of the stack.

Stacks

- Figure illustrates a stack with several nodes—stackPtr points to the stack's top element.
- Stacks and linked lists are represented identically.
- The difference between stacks and linked lists is that insertions and deletions may occur anywhere in a linked list, but only at the top of a stack.



Stacks

- The primary functions used to manipulate a stack are push and pop.
- Function push creates a new node and places it on top of the stack.
- Function pop removes a node from the top of the stack, frees the memory that was allocated to the popped node and returns the popped value.

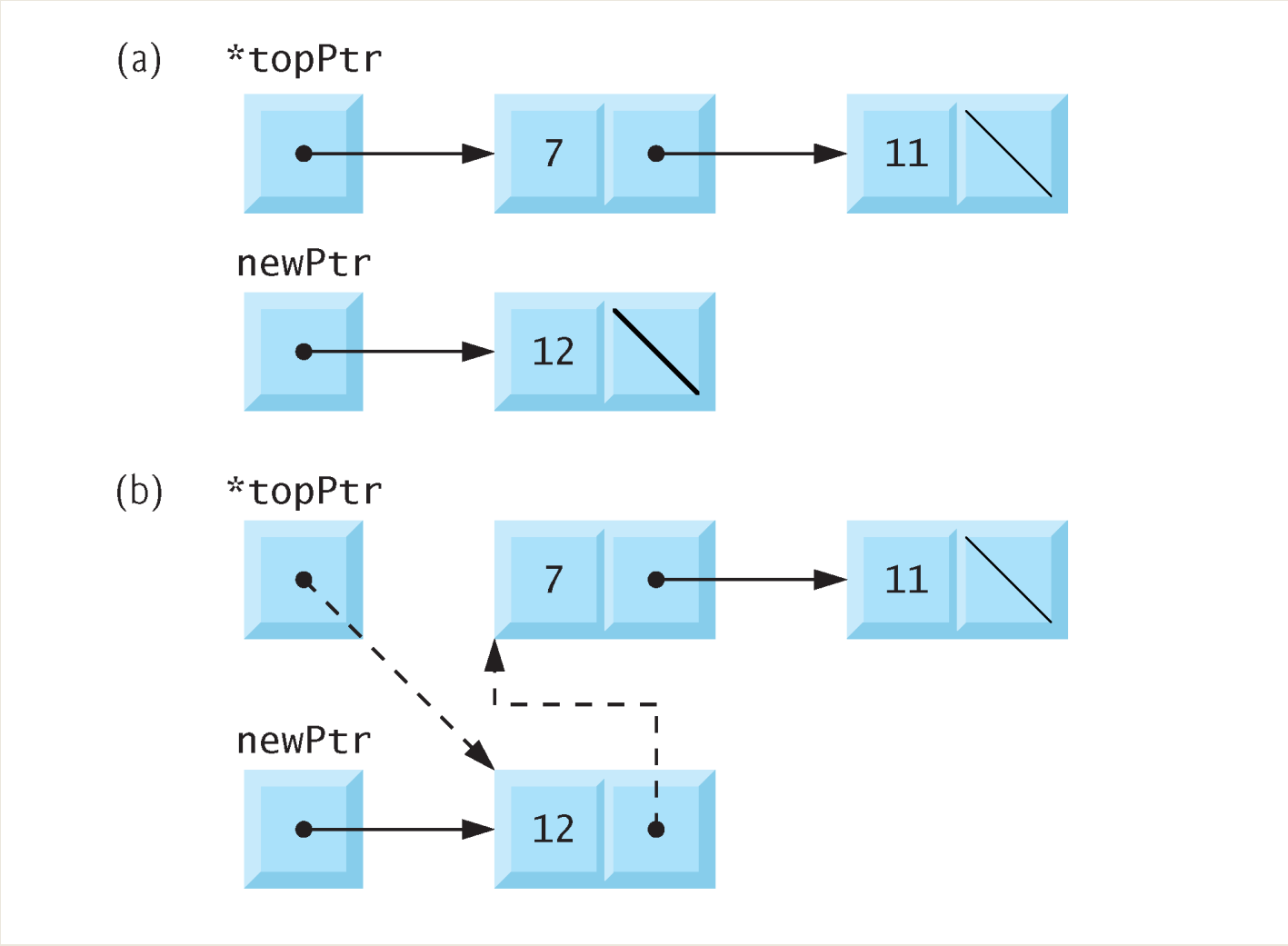
Stacks

- Example program implements a simple stack of integers.
- The program provides three options:
  - *push a value onto the stack (function push),*
  - *pop a value off the stack (function pop)*
  - *terminate the program.*

Function push

- Function push places a new node at the top of the stack.
- The function consists of three steps:
  - *Create a new node by calling malloc and assign the location of the allocated memory to newPtr.*
  - *Assign to newPtr->data the value to be placed on the stack and assign *topPtr (the stack top pointer) to newPtr->nextPtr—the link member of newPtr now points to the previous top node.*
  - *Assign newPtr to *topPtr—*topPtr now points to the new stack top.*

Function push



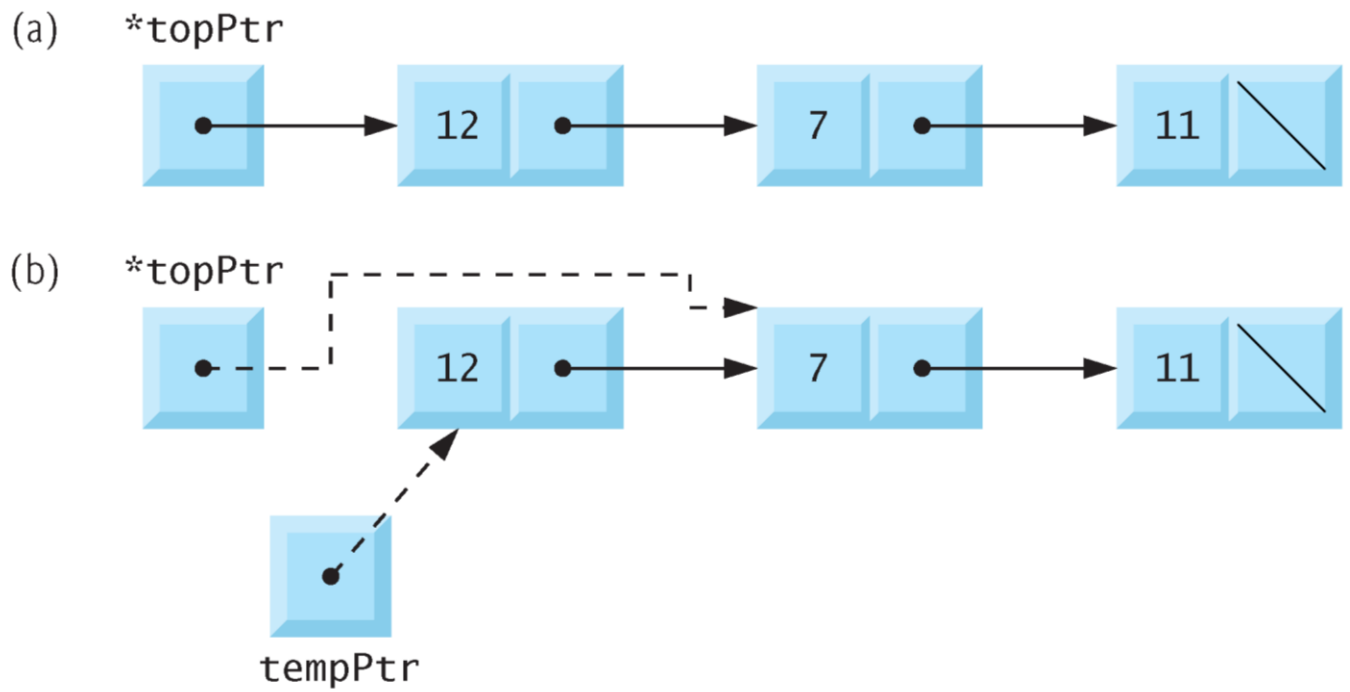
Function push

- Figure illustrates function push.
- Part (a) of the figure shows the stack and the new node before the push operation.
- The dotted arrows in part (b) illustrate Steps 2 and 3 of the push operation that enable the node containing 12 to become the new stack top.
- Manipulations involving *topPtr change the value of stackPtr in main.

Function pop

- Function pop removes a node from the top of the stack.
- Function main determines if the stack is empty before calling pop.
- The pop operation consists of five steps:
  - *Assign *topPtr to tempPtr, which will be used to free the unneeded memory*
  - *Assign (*topPtr)->data to popValue to save the value in the top node*
  - *Assign (*topPtr)->nextPtr to *topPtr so *topPtr contains address of the new top node*
  - *Free the memory pointed to by tempPtr*
  - *Return popValue to the caller*

Function pop



Function pop

- Figure illustrates function pop.
- Part (a) shows the stack after the previous push operation.
- Part (b) shows tempPtr pointing to the first node of the stack and topPtr pointing to the second node of the stack.
- Function free is used to free the memory pointed to by tempPtr.

Queues

- Another common data structure is the queue.
- A queue is similar to a checkout line in a grocery store—the first person in line is serviced first, and other customers enter the line only at the end and wait to be serviced.
- Queue nodes are removed only from the head of the queue and are inserted only at the tail of the queue.
- For this reason, a queue is referred to as a first-in, first-out (FIFO) data structure.
- The insert and remove operations are known as enqueue and dequeue, respectively.

Queues

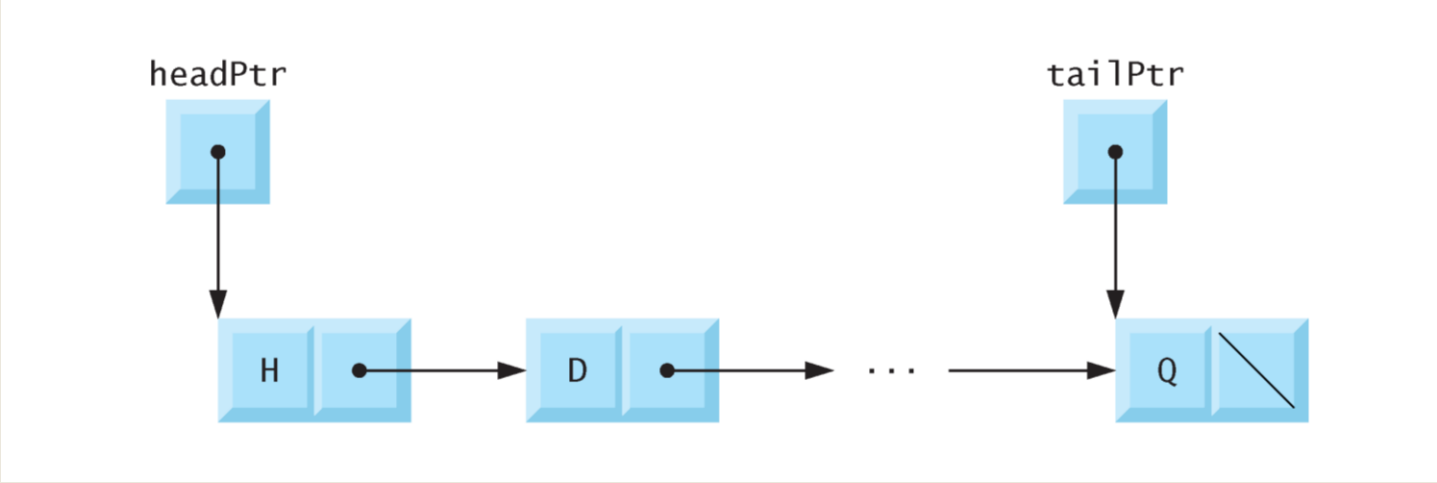
- Queues have many applications in computer systems.
- For computers that have only a single processor, only one user at a time may be serviced.
- Entries for the other users are placed in a queue.
- Each entry gradually advances to the front of the queue as users receive service.
- The entry at the front of the queue is the next to receive service.

Queues

- Queues are also used to support print spooling.
- A multiuser environment may have only a single printer.
- Many users may be generating outputs to be printed.
- If the printer is busy, other outputs may still be generated.
- These are spooled to disk where they wait in a queue until the printer becomes available.

Queues

- Figure illustrates a queue with several nodes.
- Note the pointers to the head of the queue and the tail of the queue.



Queues

- Example program performs queue manipulations.
- The program provides several options: insert a node in the queue (function enqueue), remove a node from the queue (function dequeue) and terminate the program.

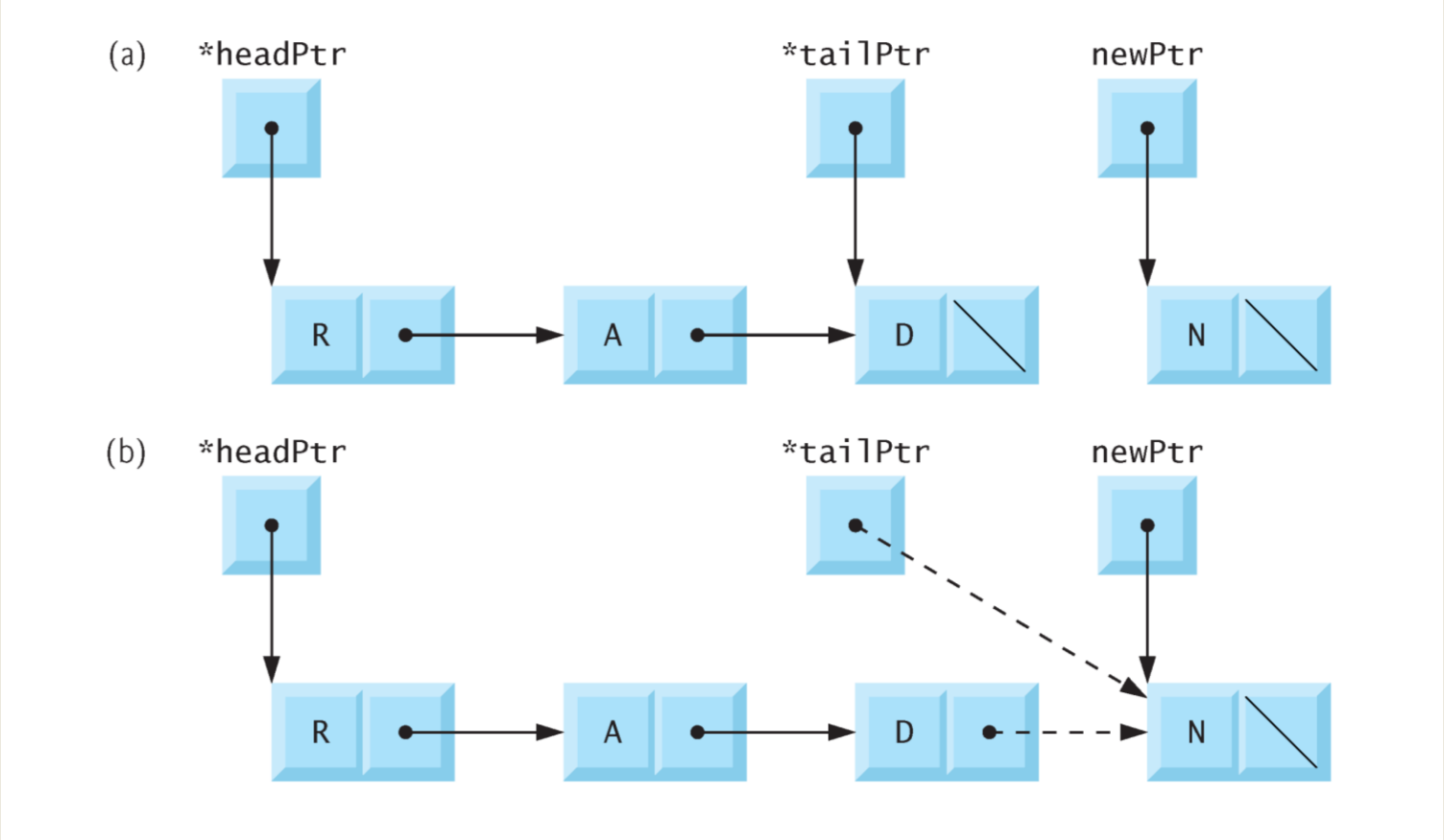
Function enqueue

- Function enqueue receives three arguments from main: the address of the pointer to the head of the queue, the address of the pointer to the tail of the queue and the value to be inserted in the queue.

Function enqueue

- The function consists of three steps:
  - *To create a new node: Call malloc, assign the allocated memory location to newPtr, assign the value to be inserted in the queue to newPtr->data and assign NULL to newPtr->nextPtr*
  - *If the queue is empty, assign newPtr to *headPtr, because the new node will be both the head and tail of the queue; otherwise, assign pointer newPtr to (*tailPtr)->nextPtr, because the new node will be placed after the previous tail node.*
  - *Assign newPtr to *tailPtr, because the new node is the queue's tail.*

Function enqueue



Function enqueue

- Figure illustrates an enqueue operation.
- Part (a) shows the queue and the new node before the operation.
- The dotted arrows in part (b) illustrate Steps 2 and 3 of function enqueue that enable a new node to be added to the end of a queue that is not empty.

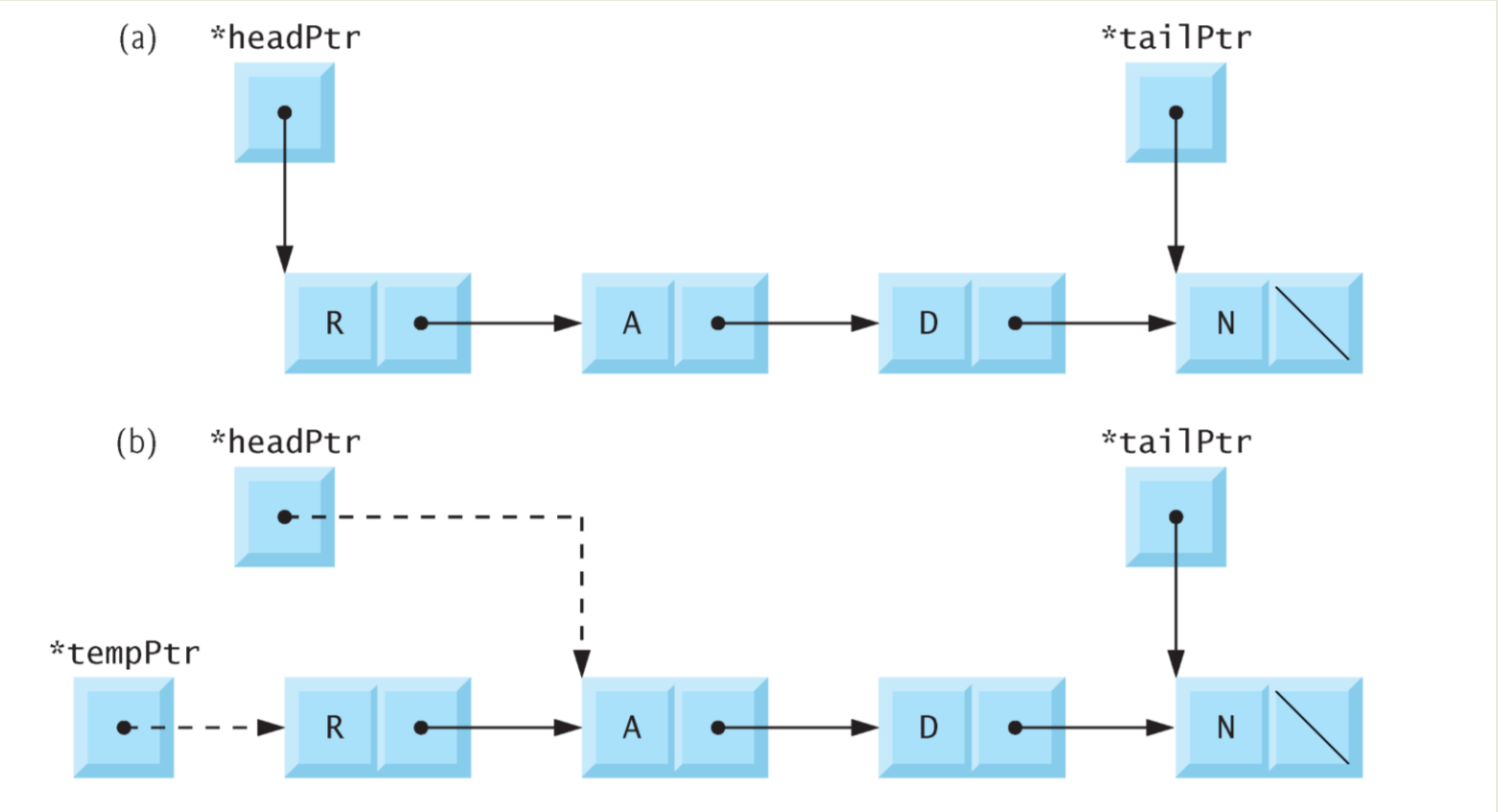
Function dequeue

- Function dequeue receives the address of the pointer to the head of the queue and the address of the pointer to the tail of the queue as arguments and removes the first node from the queue.

Function dequeue

- The dequeue operation consists of six steps:
  - *Assign (*headPtr)->data to value to save the data*
  - *Assign *headPtr to tempPtr, which will be used to free the unneeded memory*
  - *Assign (*headPtr)->nextPtr to *headPtr so that *headPtr now points to the new first node in the queue*
  - *If *headPtr is NULL, assign NULL to *tailPtr because the queue is now empty.*
  - *Free the memory pointed to by tempPtr*
  - *Return value to the caller*

Function dequeue



Function dequeue

- Figure illustrates function dequeue.
- Part (a) shows the queue after the preceding enqueue operation.
- Part (b) shows tempPtr pointing to the dequeued node, and headPtr pointing to the new first node of the queue.
- Function free is used to reclaim the memory pointed to by tempPtr.

References

- C How to Program, 8ed, by Paul Deitel and Harvey Deitel, Pearson, 2016.