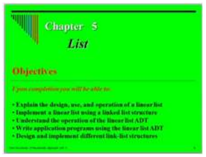


Chapter 5

List

Objectives

Upon completion you will be able to:

- Explain the design, use, and operation of a linear list
- Implement a linear list using a linked list structure
- Understand the operation of the linear list ADT
- Write application programs using the linear list ADT
- Design and implement different link-list structures

CS 1037
Computer Science Fundamentals II

Part Eleven:

Lists

5-1 Basic Operations

We begin with a discussion of the basic list operations. Each operation is developed using before and after figures to show the changes.

- Insertion
- Deletion
- Retrieval
- Traversal

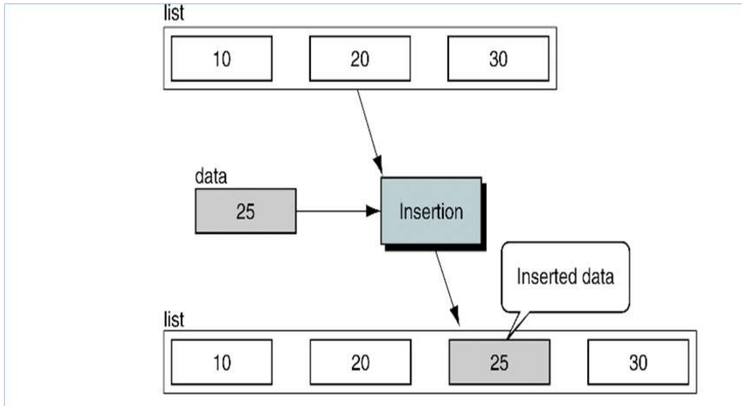


FIGURE 5-1 Insertion

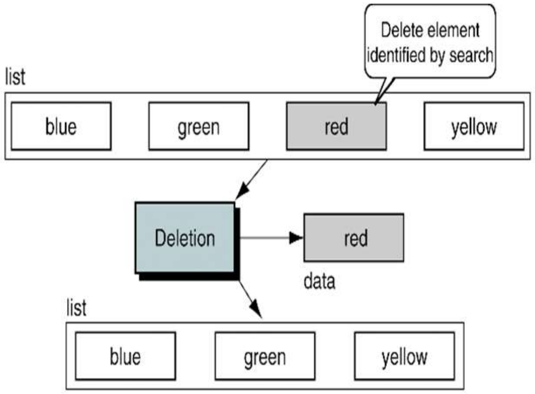


FIGURE 5-2 Deletion

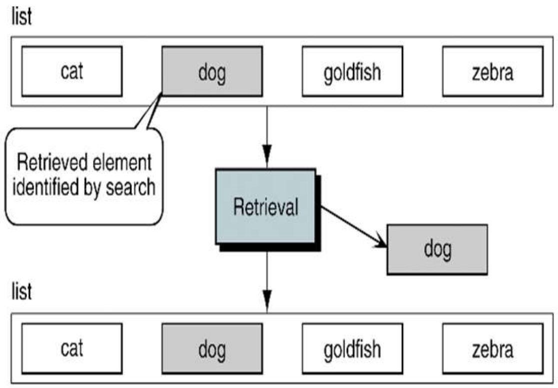


FIGURE 5-3 Retrieval

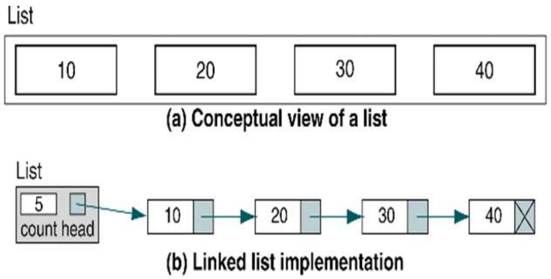
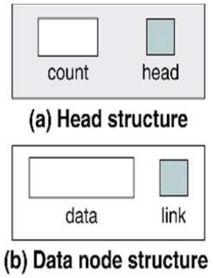


FIGURE 5-4 Linked List Implementation of a List



```
list
  count
  head
end list

node
  data
  link
end node
```

FIGURE 5-5 Head Node and Data Node

```
allocate (list)
set list head to null
set list count to 0
```

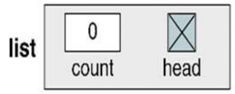
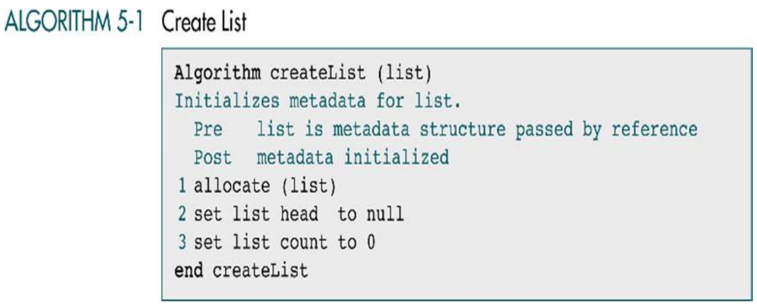


FIGURE 5-6 Create List



ALGORITHM 5-1 Create List

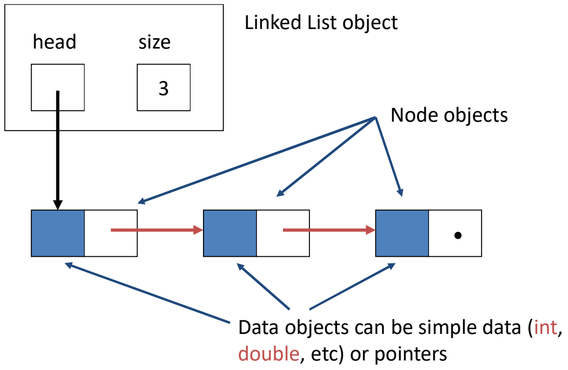
```
Algorithm createList (list)
Initializes metadata for list.
  Pre   list is metadata structure passed by reference
  Post  metadata initialized
 allocate (list)
 set list head  to null
 set list count to 0
end createList
```

5-2 Implementation

We begin with a discussion of the data structure required to support a list. We then develop the 10 basic algorithms required to build and use a list. In developing the insertion and deletion algorithms, we use extensive examples to demonstrate the analytical steps used in developing algorithms.

- Data Structure
- Algorithms

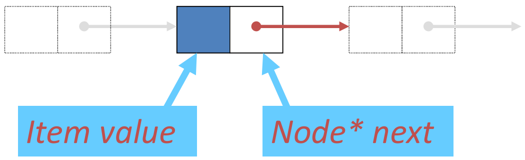
Singly Linked List



Data objects can be simple data (int, double, etc) or pointers

Nodes in Singly Linked Lists

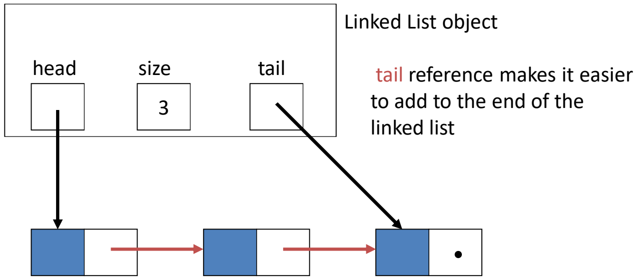
- Nodes for our linked lists will be objects dynamically created or deleted in the HEAP
  - Each *Node* object in a singly linked list will contain two member variables:



Singly Linked List

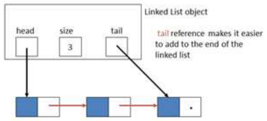
- To find the nth item in a linked list:
  - Follow the head pointer to the first node
  - Find the reference to the next node
  - Follow it to the second node
  - Find the reference to the next node
  - Follow it to the third node
  - Etc, until nth node is reached
  - The nth item is the data item in this node

Singly Linked List Variation



tail reference makes it easier to add to the end of the linked list

```
#include <stdio.h>
#include <stdlib.h>
#include "linkedList.h"

int main (void)
{
// Local Definitions
   char* dataPtr ;

   ...

   return 0;
}         // main
```



tail reference makes it easier to add to the end of the linked list

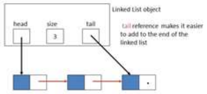
queues.h

```
#include "P4-01.h"                      /* Queue ADT Data Structures */

//             Prototype Declarations
  QUEUE* createQueue  (void);
  bool  dequeue    (QUEUE* queue, void** itemPtr);
  bool  enqueue    (QUEUE* queue, void*  itemPtr);
  bool  emptyQueue (QUEUE* queue);
  int   queueCount (QUEUE* queue);
  void printQueue (QUEUE* queue);
  bool  queueFront (QUEUE* queue, void** itemPtr);
  bool  queueRear  (QUEUE* queue, void** itemPtr);
  int   queueCount (QUEUE* queue);
  QUEUE* destroyQueue (QUEUE* queue);
  bool  fullQueue  (QUEUE* queue);

#include "P4-02.h"                      /* Create Queue */
#include "P4-03.h"                      /* Enqueue */
#include "P4-04.h"                      /* Dequeue */
#include "P4-05.h"                      /* Queue Front */
#include "P4-06.h"                      /* Queue Rear */
#include "P4-07.h"                      /* Empty Queue */
#include "P4-08.h"                      /* Full Queue */
#include "P4-09.h"                      /* Queue Count */
#include "P4-10.h"                      /* Destroy Queue */
#include "P4-14a.h"                /* Print Queue */
```

queues.h

```
#include "P4-01.h"                      /* Queue ADT Data Structures */

//             Prototype Declarations
  QUEUE* createQueue  (void);
  bool  dequeue    (QUEUE* queue, void** itemPtr);
  bool  enqueue    (QUEUE* queue, void*  itemPtr);
  bool  emptyQueue (QUEUE* queue);
  int   queueCount (QUEUE* queue);
  void printQueue (QUEUE* queue);
  bool  queueFront (QUEUE* queue, void** itemPtr);
  bool  queueRear  (QUEUE* queue, void** itemPtr);
  int   queueCount (QUEUE* queue);
  QUEUE* destroyQueue (QUEUE* queue);
  bool  fullQueue  (QUEUE* queue);

#include "P4-02.h"                      /* Create Queue */
#include "P4-03.h"                      /* Enqueue */
#include "P4-04.h"                      /* Dequeue */
#include "P4-05.h"                      /* Queue Front */
#include "P4-06.h"                      /* Queue Rear */
#include "P4-07.h"                      /* Empty Queue */
#include "P4-08.h"                      /* Full Queue */
#include "P4-09.h"                      /* Queue Count */
#include "P4-10.h"                      /* Destroy Queue */
#include "P4-14a.h"                      /* Print Queue */
```

```
#include <stdio.h>
#include <stdlib.h>
#include "linkedList.h"

int main (void)
{
// Local Definitions
   char* dataPtr ;

   LIST*  sList;
   ...

   return 0;
}         // main
```


Linked List object
head
size
3
tail
tail reference makes it easier to add to the end of the linked list

P4-01.h

```
// List ADT Type Defintions
   typedef struct node
   {
      void*          dataPtr;
      struct node*   link;
   } NODE;

   typedef struct
   {
      int       count;
      NODE*     head;
      NODE*     tail;
   } LIST;
```

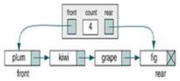
```
#include <stdio.h>
#include <stdlib.h>
#include "linkedList.h"

int main (void)
{
// Local Definitions
   char* dataPtr ;
   LIST*  sList;

   sList = createList();
   ...
   return 0;
}         // main
```

P4-02.h

```
LIST* createList(void)
{
   LIST* list;
   list= (LIST*) malloc (sizeof (LIST));
       if (list)
          {
            list->head = NULL;
            list->tail= NULL;
            list->count = 0
          } // if
       return list;
}       // createList
```

Label	Address	Value
dataPtr	326 - 329	
sList	400 - 403	
	…	
list	510 -513	10100
	…	
	…	
head	10100 - 10103	NULL
tail	10104 - 10107	NULL
count	10108 - 10111	0
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	

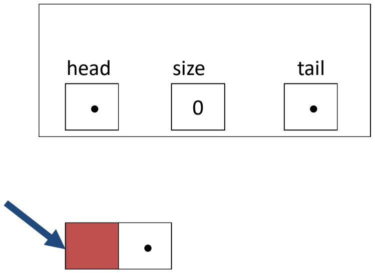
```
   ...
for (int i = 1; i<=3; i++)
   {
     dataPtr = (char*) malloc (sizeof(char));
     *dataPtr = 64 + i;
     enqueue (ourQ, dataPtr);
   }
   ...
}         // main
```



P5-04b.h

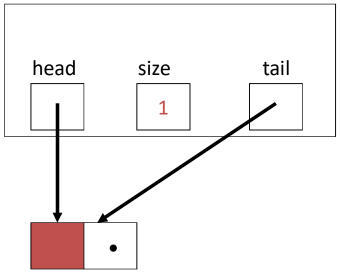
```
bool insertList (LIST* list, void* itemPtr)
{
   NODE* newPtr;
   if (!(newPtr =
       (NODE*)malloc(sizeof(NODE))))
     return false;

   newPtr->dataPtr = itemPtr;
   newPtr->link    = list->head;

   if (list->count == 0)
      list->rear = newPtr;

   (list->count)++;
   list->head = newPtr;
   return true;
}         // insertList
```

To Add an Item to an Empty Linked List


head
size
tail
0

Build the new node, and put the new data item in it

To Add an Item to an Empty Linked List



Make ***both*** head and tail point at the new node, and increment the list's size

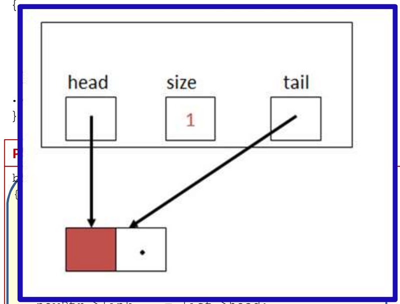
```
    ...
 for (int i = 1; i<=3; i++)
{
   dataPtr = (char*) malloc (sizeof(char));
   *dataPtr = 64 + i;
   insertListEND(sList, dataPtr);
 }
...
}       // main
```

P5-04b.h

```
bool insertList (LIST* list, void* itemPtr)
{
   NODE* newPtr;
   if (!(newPtr =
      (NODE*)malloc(sizeof(NODE))))
     return false;

   newPtr->dataPtr = itemPtr;
   newPtr->link    = list->head;

   if (list->count == 0)
      list->rear = newPtr;

   (list->count)++;
   list->head = newPtr;
   return true;
}         // insertList
```

Label	Address	Value
dataPtr	326 - 329	
sList	400 - 403	10100
	…	
	…	
	…	
	…	
head	10100 - 10103	NULL
tail	10104 - 10107	NULL
count	10108 - 10111	0
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	

```
    ...
 for (int i = 1; i<=3; i++)
{
                                                  ;
.
}
```



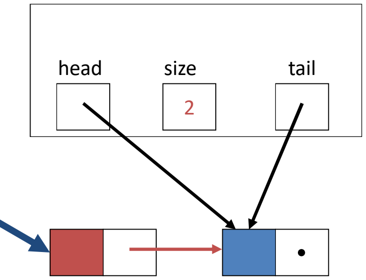
```
   newPtr->link    = list->head;

   if (list->count == 0)
      list->rear = newPtr;

   (list->count)++;
   list->head = newPtr;
   return true;
}         // insertList
```

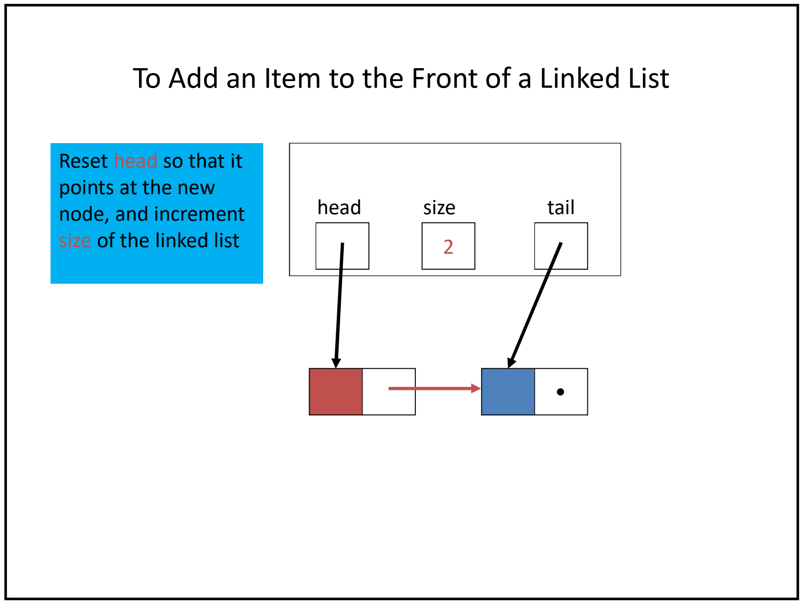
Label	Address	Value
dataPtr	326 - 329	10210
sList	400 - 403	10100
i	404 - 407	1
list	420 - 423	10100
itemPtr	424 - 427	10210
	…	
head	10100 - 10103	10400
tail	10104 - 10107	10400
count	10108 - 10111	1
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	NULL
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	

To Add an Item to the Front of a Linked List

Build the new node, and put the new data item in it,

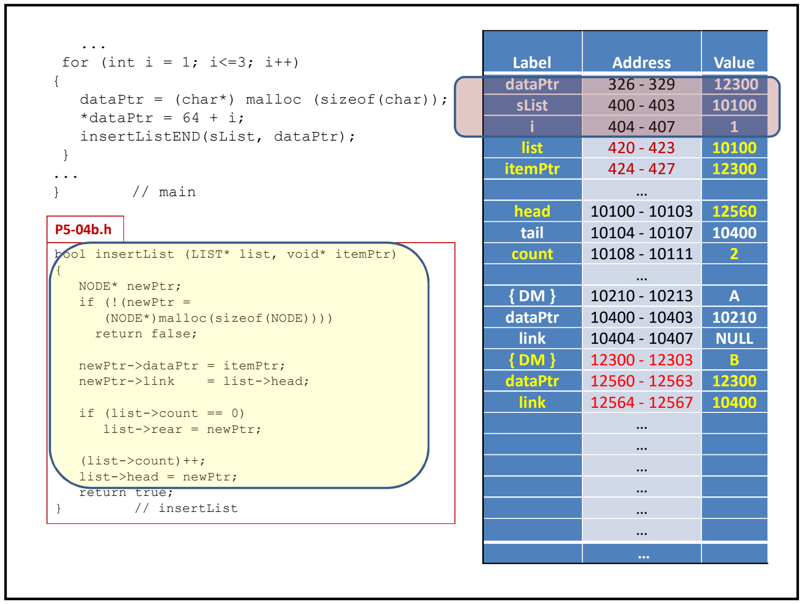
make it point an the current front node





To Add an Item to the Front of a Linked List

Reset head so that it points at the new node, and increment size of the linked list

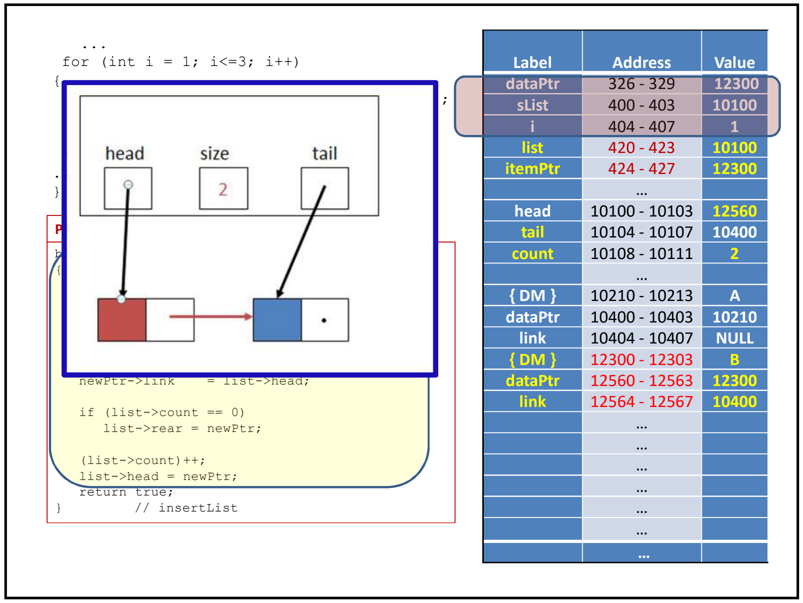


```
   ...
 for (int i = 1; i<=3; i++)
{
   dataPtr = (char*) malloc (sizeof(char));
   *dataPtr = 64 + i;
   insertListEND(sList, dataPtr);
 }
...
}        // main
```

P5-04b.h

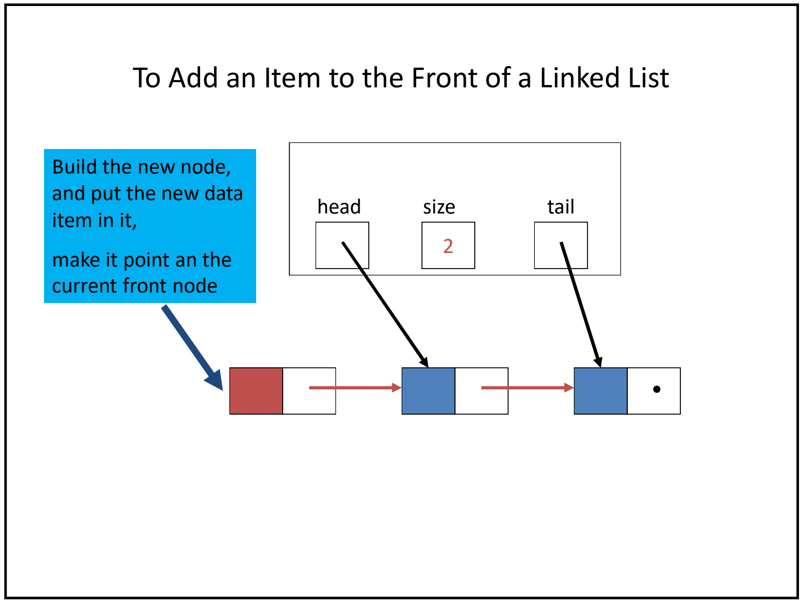
```
bool insertList (LIST* list, void* itemPtr)
{
   NODE* newPtr;
   if (!(newPtr =
      (NODE*)malloc(sizeof(NODE))))
     return false;

   newPtr->dataPtr = itemPtr;
   newPtr->link    = list->head;

   if (list->count == 0)
      list->rear = newPtr;

   (list->count)++;
   list->head = newPtr;
   return true;
}          // insertList
```

Label	Address	Value
dataPtr	326 - 329	12300
sList	400 - 403	10100
i	404 - 407	1
list	420 - 423	10100
itemPtr	424 - 427	12300
	…	
head	10100 - 10103	12560
tail	10104 - 10107	10400
count	10108 - 10111	2
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	NULL
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	10400
	…	
	…	
	…	
	…	
	…	
	…	
	…	



```
   ...
 for (int i = 1; i<=3; i++)
{
   newPtr->link    = list->head;

   if (list->count == 0)
      list->rear = newPtr;

   (list->count)++;
   list->head = newPtr;
   return true;
}          // insertList
```

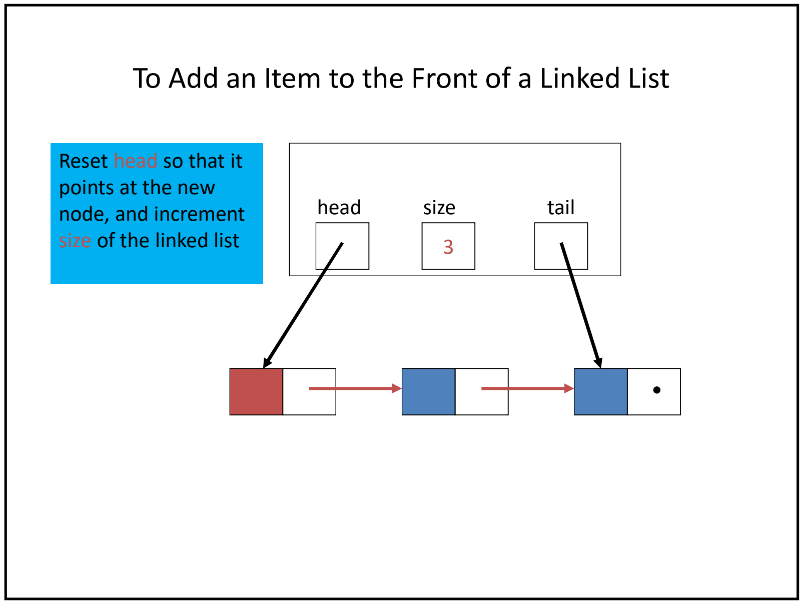
Label	Address	Value
dataPtr	326 - 329	12300
sList	400 - 403	10100
i	404 - 407	1
list	420 - 423	10100
itemPtr	424 - 427	12300
	…	
head	10100 - 10103	12560
tail	10104 - 10107	10400
count	10108 - 10111	2
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	NULL
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	10400
	…	
	…	
	…	
	…	
	…	
	…	
	…	



To Add an Item to the Front of a Linked List

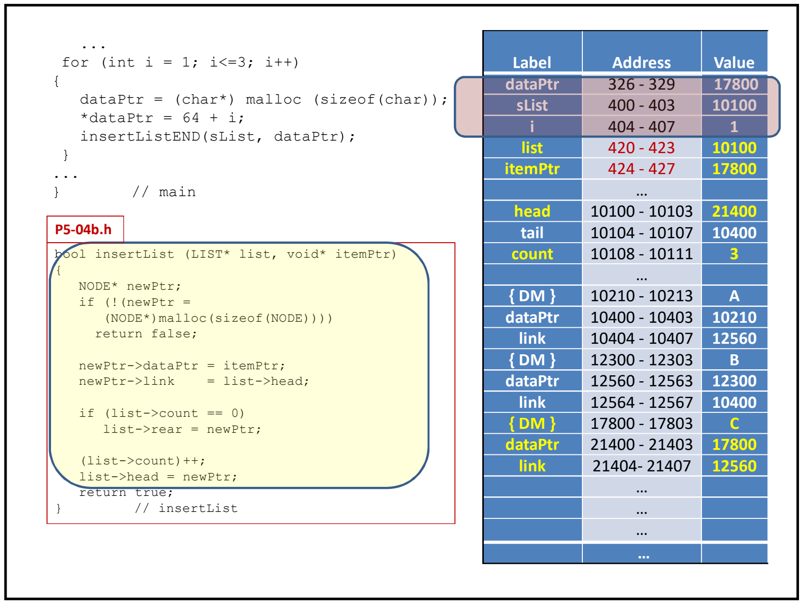
Build the new node, and put the new data item in it,

make it point an the current front node



To Add an Item to the Front of a Linked List

Reset head so that it points at the new node, and increment size of the linked list

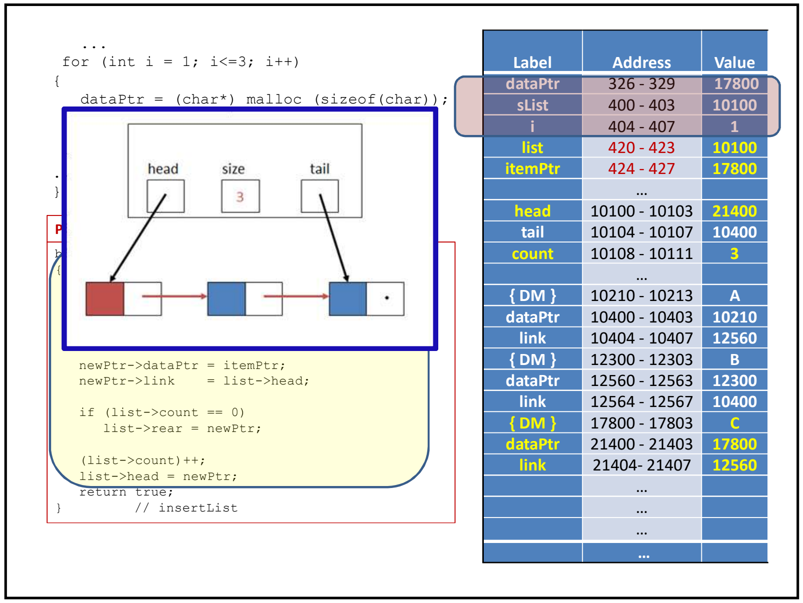


```
   ...
 for (int i = 1; i<=3; i++)
{
   dataPtr = (char*) malloc (sizeof(char));
   *dataPtr = 64 + i;
   insertListEND(sList, dataPtr);
 }
...
}        // main
```

P5-04b.h

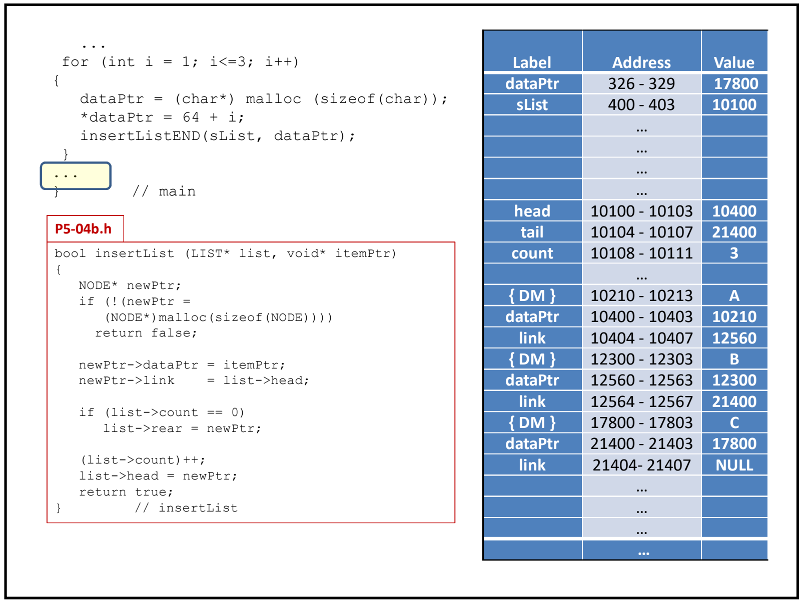
```
bool insertList (LIST* list, void* itemPtr)
{
   NODE* newPtr;
   if (!(newPtr =
      (NODE*)malloc(sizeof(NODE))))
     return false;

   newPtr->dataPtr = itemPtr;
   newPtr->link    = list->head;

   if (list->count == 0)
      list->rear = newPtr;

   (list->count)++;
   list->head = newPtr;
   return true;
}          // insertList
```

Label	Address	Value
dataPtr	326 - 329	17800
sList	400 - 403	10100
i	404 - 407	1
list	420 - 423	10100
itemPtr	424 - 427	17800
	…	
head	10100 - 10103	21400
tail	10104 - 10107	10400
count	10108 - 10111	3
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	10400
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
link	21404- 21407	12560
	…	
	…	
	…	
	…	



```
   ...
 for (int i = 1; i<=3; i++)
{
   dataPtr = (char*) malloc (sizeof(char));
.
}
P
b
{
   newPtr->dataPtr = itemPtr;
   newPtr->link    = list->head;

   if (list->count == 0)
      list->rear = newPtr;

   (list->count)++;
   list->head = newPtr;
   return true;
}          // insertList
```

Label	Address	Value
dataPtr	326 - 329	17800
sList	400 - 403	10100
i	404 - 407	1
list	420 - 423	10100
itemPtr	424 - 427	17800
	…	
head	10100 - 10103	21400
tail	10104 - 10107	10400
count	10108 - 10111	3
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	10400
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
link	21404- 21407	12560
	…	
	…	
	…	
	…	



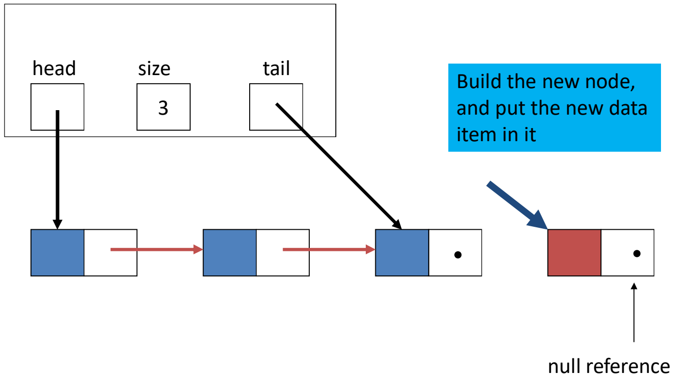
```
   ...
 for (int i = 1; i<=3; i++)
{
   dataPtr = (char*) malloc (sizeof(char));
   *dataPtr = 64 + i;
   insertListEND(sList, dataPtr);
 }
...
}        // main
```

P5-04b.h

```
bool insertList (LIST* list, void* itemPtr)
{
   NODE* newPtr;
   if (!(newPtr =
      (NODE*)malloc(sizeof(NODE))))
     return false;

   newPtr->dataPtr = itemPtr;
   newPtr->link    = list->head;

   if (list->count == 0)
      list->rear = newPtr;

   (list->count)++;
   list->head = newPtr;
   return true;
}          // insertList
```

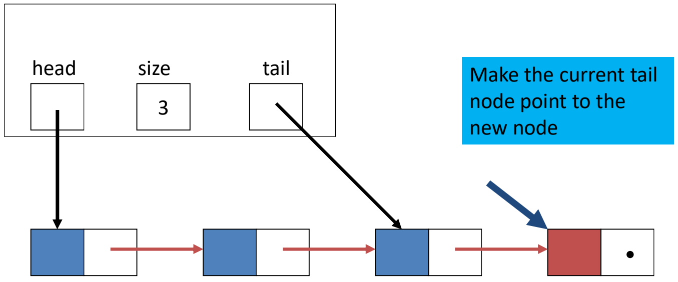
Label	Address	Value
dataPtr	326 - 329	17800
sList	400 - 403	10100
	…	
	…	
	…	
	…	
head	10100 - 10103	10400
tail	10104 - 10107	21400
count	10108 - 10111	3
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	21400
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
link	21404- 21407	NULL
	…	
	…	
	…	
	…	

To Add an Item to the End of a Linked List



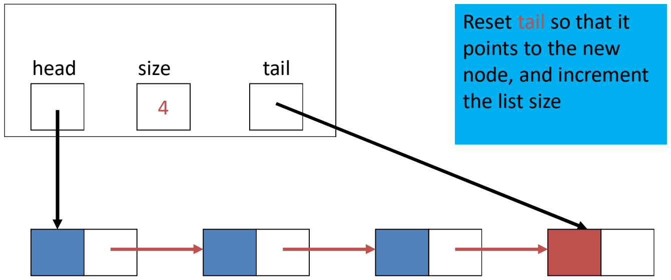
Build the new node, and put the new data item in it

To Add an Item to the End of a Linked List



Make the current tail node point to the new node

To Add an Item to the End of a Linked List



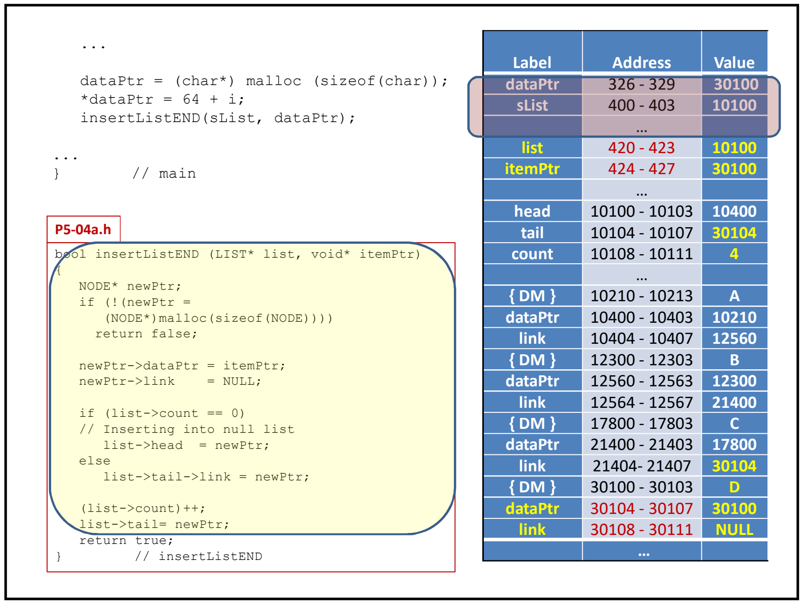
Reset tail so that it points to the new node, and increment the list size

```
   . . .

   dataPtr = (char*) malloc (sizeof(char));
   *dataPtr = 64 + i;
   insertListEND(sList, dataPtr);

. . .
}         // main
```

P5-04a.h

```
bool insertListEND (LIST* list, void* itemPtr)
{
   NODE* newPtr;
   if (!(newPtr =
      (NODE*)malloc(sizeof(NODE))))
     return false;

   newPtr->dataPtr = itemPtr;
   newPtr->link    = NULL;

   if (list->count == 0)
   // Inserting into null list
      list->head  = newPtr;
   else
      list->tail->link = newPtr;

   (list->count)++;
   list->tail= newPtr;
   return true;
}         // insertListEND
```

Label	Address	Value
dataPtr	326 - 329	30100
sList	400 - 403	10100
	…	
	…	
	…	
	…	
head	10100 - 10103	10400
tail	10104 - 10107	21400
count	10108 - 10111	3
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	21400
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
link	21404- 21407	NULL
{ DM }	30100 - 30103	D
	…	
	…	
	…	

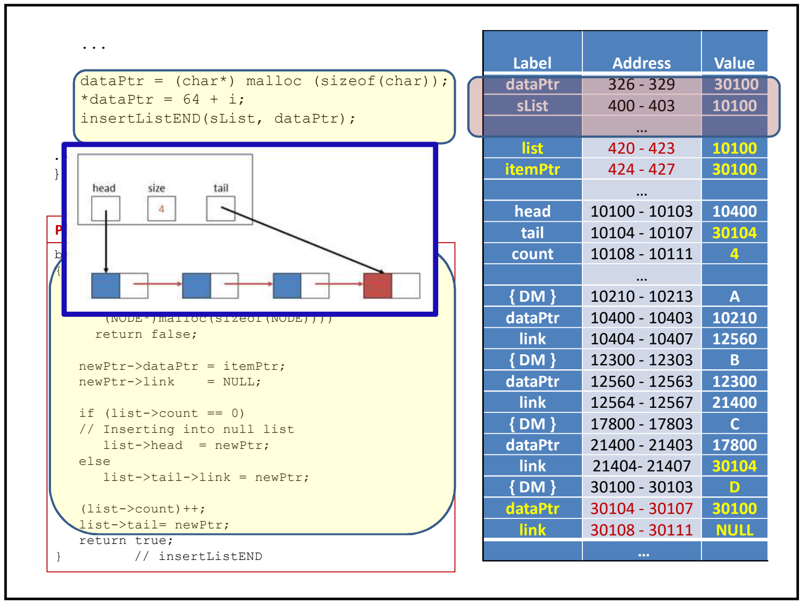


```
   ...

   dataPtr = (char*) malloc (sizeof(char));
   *dataPtr = 64 + i;
   insertListEND(sList, dataPtr);

...
}        // main
```

P5-04a.h

```
bool insertListEND (LIST* list, void* itemPtr)
{
   NODE* newPtr;
   if (!(newPtr =
      (NODE*)malloc(sizeof(NODE))))
     return false;

   newPtr->dataPtr = itemPtr;
   newPtr->link    = NULL;

   if (list->count == 0)
   // Inserting into null list
      list->head  = newPtr;
   else
      list->tail->link = newPtr;

   (list->count)++;
   list->tail= newPtr;
   return true;
}          // insertListEND
```

Label	Address	Value
dataPtr	326 - 329	30100
sList	400 - 403	10100
	…	
list	420 - 423	10100
itemPtr	424 - 427	30100
	…	
head	10100 - 10103	10400
tail	10104 - 10107	30104
count	10108 - 10111	4
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	21400
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
link	21404- 21407	30104
{ DM }	30100 - 30103	D
dataPtr	30104 - 30107	30100
link	30108 - 30111	NULL
	…	

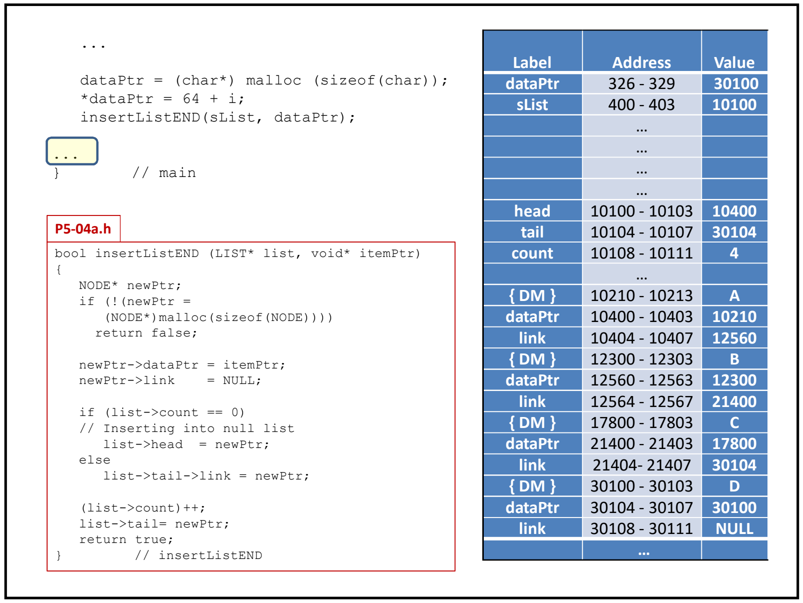


```
   ...

   dataPtr = (char*) malloc (sizeof(char));
   *dataPtr = 64 + i;
   insertListEND(sList, dataPtr);

.
}
```

P

```
b
{
      (NODE*)malloc(sizeof(NODE))))
     return false;

   newPtr->dataPtr = itemPtr;
   newPtr->link    = NULL;

   if (list->count == 0)
   // Inserting into null list
      list->head  = newPtr;
   else
      list->tail->link = newPtr;

   (list->count)++;
   list->tail= newPtr;
   return true;
}          // insertListEND
```

Label	Address	Value
dataPtr	326 - 329	30100
sList	400 - 403	10100
	…	
list	420 - 423	10100
itemPtr	424 - 427	30100
	…	
head	10100 - 10103	10400
tail	10104 - 10107	30104
count	10108 - 10111	4
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	21400
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
link	21404- 21407	30104
{ DM }	30100 - 30103	D
dataPtr	30104 - 30107	30100
link	30108 - 30111	NULL
	…	

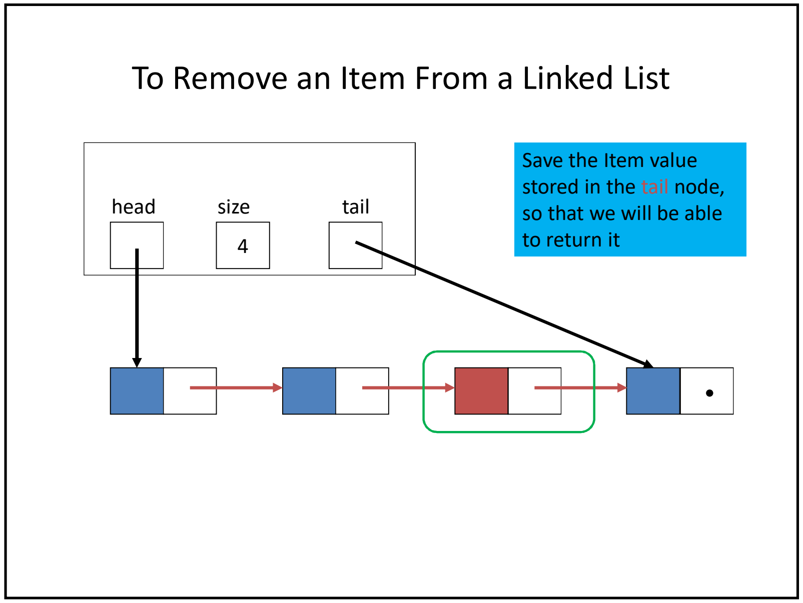


```
   ...

   dataPtr = (char*) malloc (sizeof(char));
   *dataPtr = 64 + i;
   insertListEND(sList, dataPtr);

...
}        // main
```

P5-04a.h

```
bool insertListEND (LIST* list, void* itemPtr)
{
   NODE* newPtr;
   if (!(newPtr =
      (NODE*)malloc(sizeof(NODE))))
     return false;

   newPtr->dataPtr = itemPtr;
   newPtr->link    = NULL;

   if (list->count == 0)
   // Inserting into null list
      list->head  = newPtr;
   else
      list->tail->link = newPtr;

   (list->count)++;
   list->tail= newPtr;
   return true;
}          // insertListEND
```

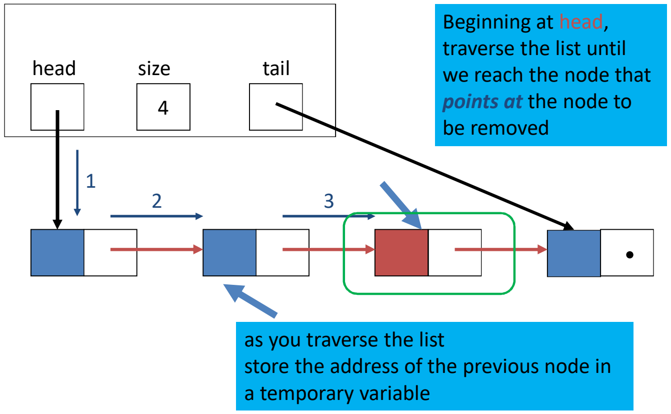
Label	Address	Value
dataPtr	326 - 329	30100
sList	400 - 403	10100
	…	
	…	
	…	
	…	
head	10100 - 10103	10400
tail	10104 - 10107	30104
count	10108 - 10111	4
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	21400
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
link	21404- 21407	30104
{ DM }	30100 - 30103	D
dataPtr	30104 - 30107	30100
link	30108 - 30111	NULL
	…	



To Remove an Item From a Linked List

Save the Item value stored in the tail node, so that we will be able to return it

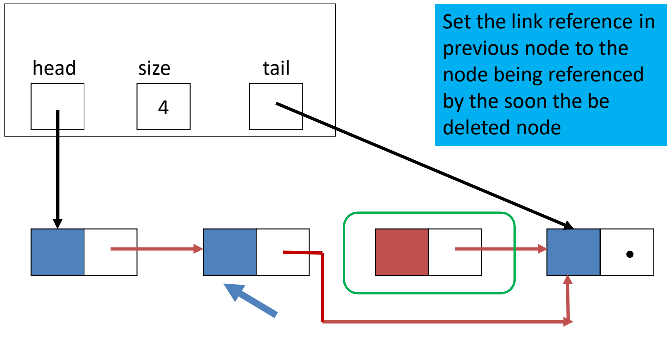
To Remove an Item From a Linked List



Beginning at head, traverse the list until we reach the node that ***points at*** the node to be removed

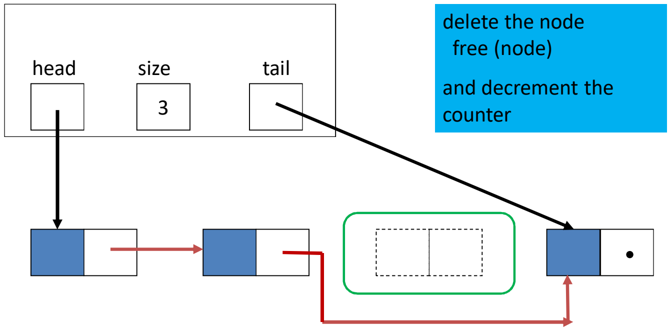
as you traverse the list
store the address of the previous node in
a temporary variable

To Remove an Item From a Linked List



Set the link reference in previous node to the node being referenced by the soon the be deleted node

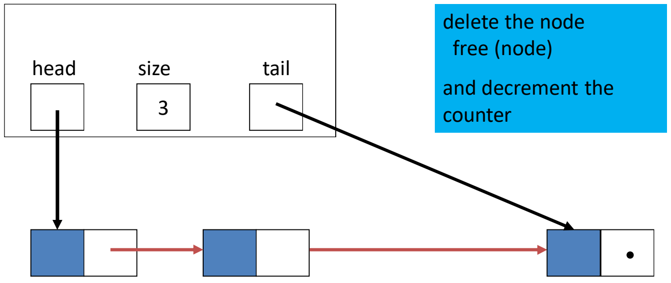
To Remove an Item From a Linked List



delete the node
free (node)

and decrement the counter

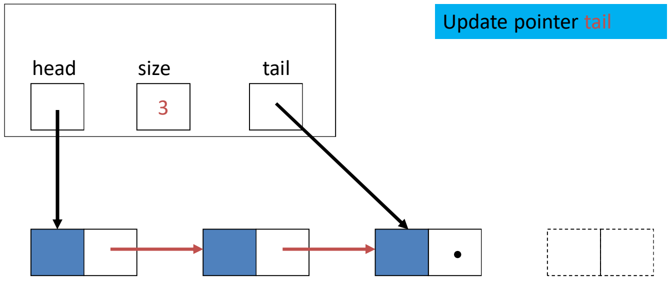
To Remove an Item From a Linked List



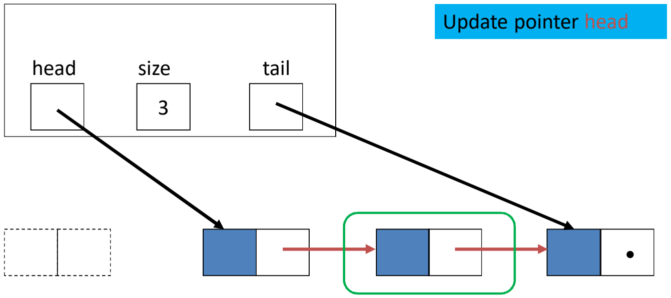
delete the node
free (node)

and decrement the counter

IF Node is Last Item From a Linked List


Update pointer tail

IF Node is First Item From a Linked List


Update pointer head

```
    ...
    deleteNode(sList, 3);
    ...
}         // main
```

P5-07a.h

```
int deleteNode(LIST* list, int pos)  {
    NODE* tempNode;
    NODE* preNode;
    if (pos > list->count)
        return 0;
    tempNode = list->head;
    preNode = list->head;
    if (pos == 1)
      list->head = tempNode->link;
    else
    {
        for (int i = 1; i < pos; i++)
        {
           preNode = tempNode;
           tempNode = tempNode->link;
        }
        preNode->link = tempNode->link;
        if (pos == list->count)
        {
            list->rear = preNode;
        }
    }

    free(tempNode->dataPtr);
    free(tempNode);
    list->count--;
    return 1;
}
```

Label	Address	Value
dataPtr	326 - 329	30100
sList	400 - 403	10100
	…	
	…	
	…	
	…	
head	10100 - 10103	10400
tail	10104 - 10107	30104
count	10108 - 10111	4
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	21400
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
link	21404- 21407	30104
{ DM }	30100 - 30103	D
dataPtr	30104 - 30107	30100
link	30108 - 30111	NULL
	…	

```
    ...
    deleteNode(sList, 3);
    ...
}         // main
```

P5-07a.h

```
int deleteNode(LIST* list, int pos)  {
    NODE* tempNode;
    NODE* preNode;
    if (pos > list->count)
        return 0;
    tempNode = list->head;
    preNode = list->head;
    if (pos == 1)
      list->head = tempNode->link;
    else
    {
        for (int i = 1; i < pos; i++)
        {
           preNode = tempNode;
           tempNode = tempNode->link;
        }
        preNode->link = tempNode->link;
        if (pos == list->count)
        {
            list->rear = preNode;
        }
    }

    free(tempNode->dataPtr);
    free(tempNode);
    list->count--;
    return 1;
}
```

Label	Address	Value
dataPtr	326 - 329	30100
sList	400 - 403	10100
list	512 - 515	10100
pos	516 - 519	3
tempNode	520 - 523	21400
preNode	524 -527	12560
head	10100 - 10103	10400
tail	10104 - 10107	30104
count	10108 - 10111	3
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	30104
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
link	21404- 21407	30104
{ DM }	30100 - 30103	D
dataPtr	30104 - 30107	30100
link	30108 - 30111	NULL
	…	

```
    ...
    deleteNode(sList, 3);
    ...
}           // main
```

P5-07a.h

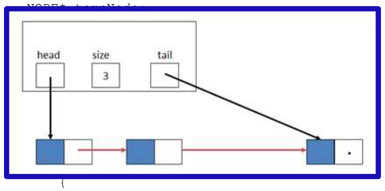
```
int deleteNode(LIST* list, int pos)  {
    NODE* tempNode;
    NODE* preNode;
    if (pos > list->count)
        return 0;
    tempNode = list->head;
    preNode = list->head;
    if (pos == 1)
      list->head = tempNode->link;
    else
    {
        for (int i = 1; i < pos; i++)
        {
           preNode = tempNode;
           tempNode = tempNode->link;
        }
        preNode->link = tempNode->link;
        if (pos == list->count)
        {
            list->rear = preNode;
        }
    }

    free(tempNode->dataPtr);
    free(tempNode);
    list->count--;
    return 1;
}
```

Label	Address	Value
dataPtr	326 - 329	30100
sList	400 - 403	10100
	…	
	…	
	…	
	…	
head	10100 - 10103	10400
tail	10104 - 10107	30104
count	10108 - 10111	3
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	30104
	…	
	…	
	…	
{ DM }	30100 - 30103	D
dataPtr	30104 - 30107	30100
link	30108 - 30111	NULL
	…	

```
    ...
    deleteNode(sList, 3);
    ...
}           // main
```

P5-07a.h

```
int deleteNode(LIST* list, int pos)  {
    NODE* tempNode;
```



```
        {
           preNode = tempNode;
           tempNode = tempNode->link;
        }
        preNode->link = tempNode->link;
        if (pos == list->count)
        {
            list->rear = preNode;
        }
    }

    free(tempNode->dataPtr);
    free(tempNode);
    list->count--;
    return 1;
}
```

Label	Address	Value
dataPtr	326 - 329	30100
sList	400 - 403	10100
	…	
	…	
	…	
	…	
head	10100 - 10103	10400
tail	10104 - 10107	30104
count	10108 - 10111	3
	…	
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
link	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
link	12564 - 12567	30104
	…	
	…	
	…	
{ DM }	30100 - 30103	D
dataPtr	30104 - 30107	30100
link	30108 - 30111	NULL
	…	

Linked List Operations Based on Data Objects

- It is possible to use the code that transverses a linked list structure to create a ‘search’ function.

Doubly-Linked Lists

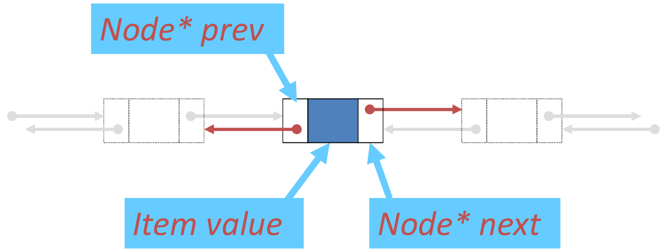
- A common variation on linked lists is to have two pointers to other nodes within each node: one to the *next* node on the list, and one to the *previous* node
- Doubly-linked lists make some operations, such as deleting a tail node, more efficient
- Doubly-linked lists can have *iterators* for efficient forward and backward traversals
  - iterator can now have *operator++* and *operator- -*

Doubly-Linked Lists

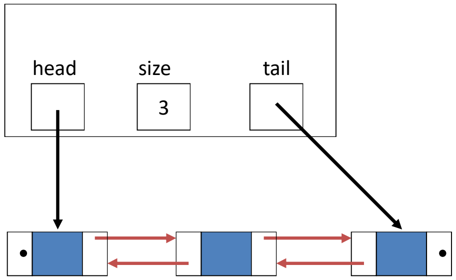
- Other operations, such as adding an item to an ordered linked list, are easier to program with doubly-linked lists
- *Tradeoffs*:
  - Each node requires 4 additional bytes

Nodes in Doubly Linked Lists

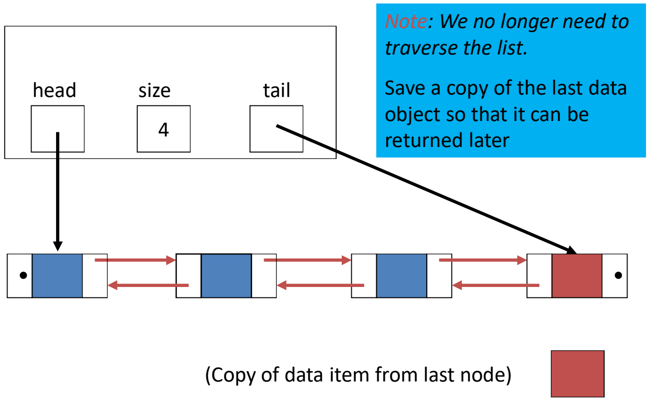
- Each *Node* object in a doubly linked list will contain three member variables:



Doubly-Linked List

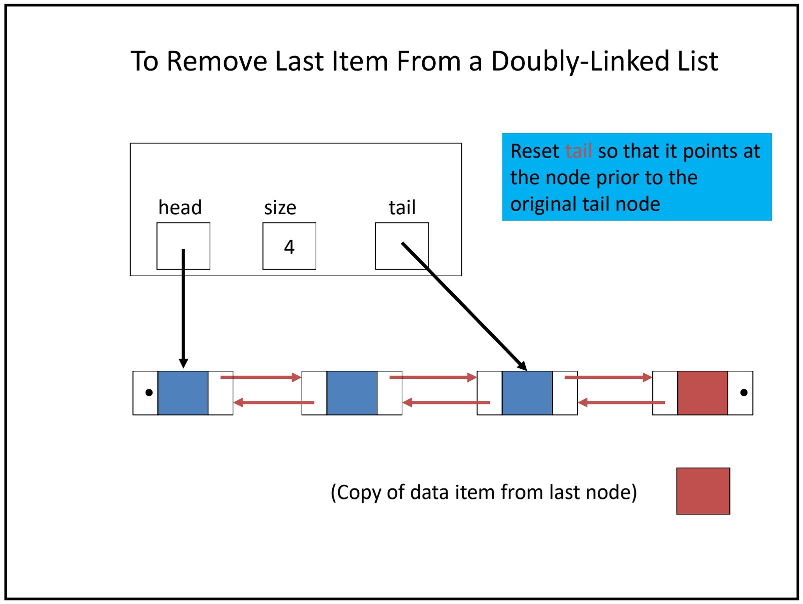


To Remove Last Item From a Doubly-Linked List



Note: We no longer need to traverse the list.

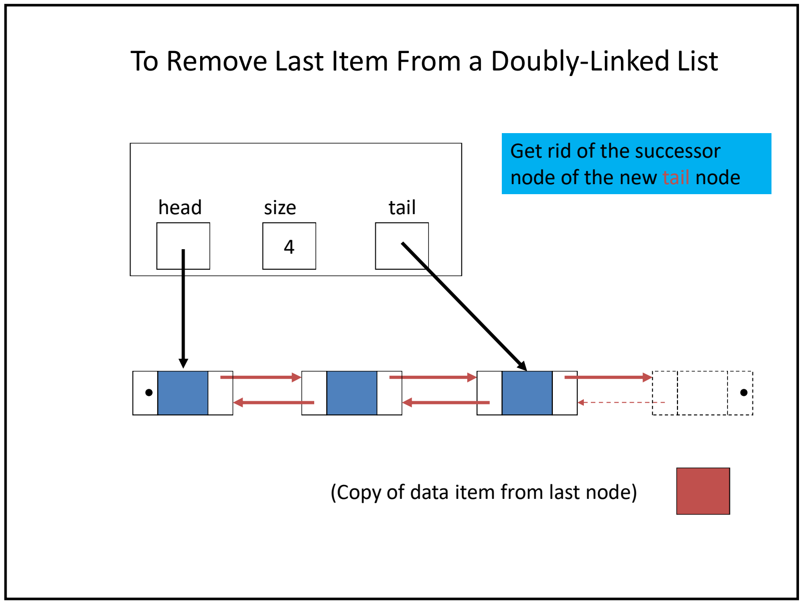
Save a copy of the last data object so that it can be returned later



To Remove Last Item From a Doubly-Linked List

Reset tail so that it points at the node prior to the original tail node

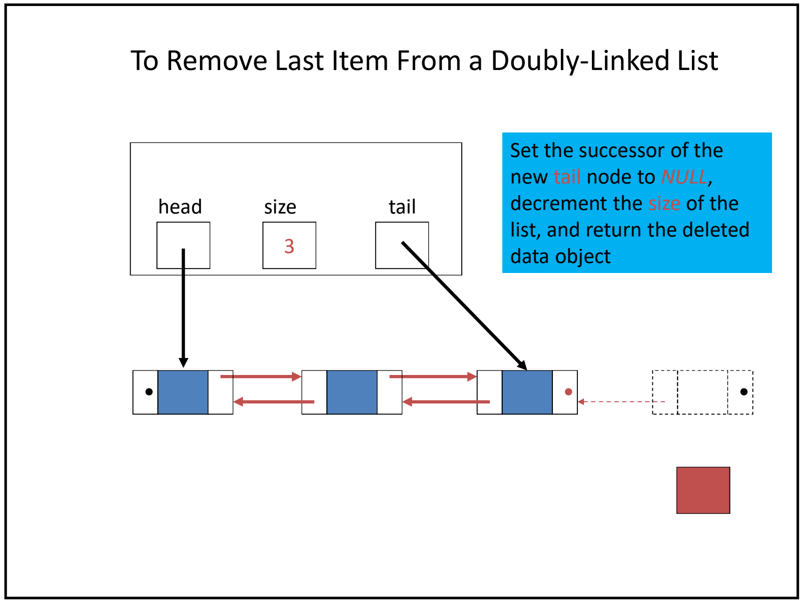
(Copy of data item from last node)



To Remove Last Item From a Doubly-Linked List

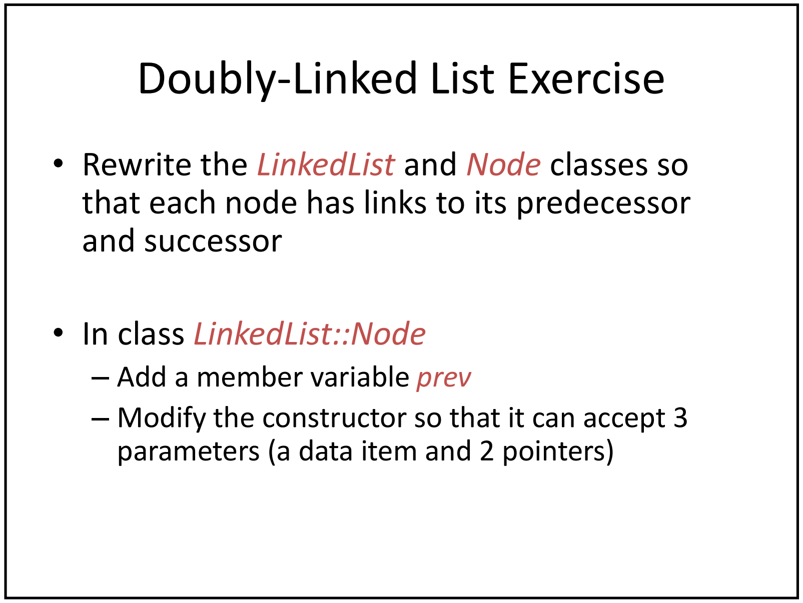
Get rid of the successor node of the new tail node

(Copy of data item from last node)



To Remove Last Item From a Doubly-Linked List

Set the successor of the new tail node to *NULL*, decrement the size of the list, and return the deleted data object



Doubly-Linked List Exercise

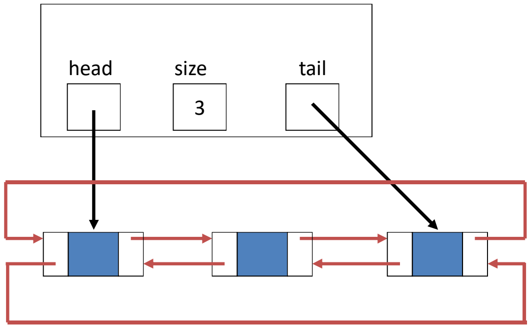
- Rewrite the *LinkedList* and *Node* classes so that each node has links to its predecessor and successor
- In class *LinkedList::Node*
  - Add a member variable *prev*
  - Modify the constructor so that it can accept 3 parameters (a data item and 2 pointers)

Circular Linked Lists

Circular Linked Lists

- Circular linked lists avoid the use of null references in their nodes
- Can be useful for certain algorithms
- Can also make programming simpler: fewer special cases to consider
- Successor of tail node is the head node; in doubly-linked list, predecessor of head node is the tail node

Circular Doubly-Linked List



Circular Doubly-Linked List Implementation

- *LinkedList::Node* class is the same as for the (non-circular) doubly-linked list implementation
- ***Exercise***: Try rewriting *LinkedList* so that it implements a circular doubly-linked list