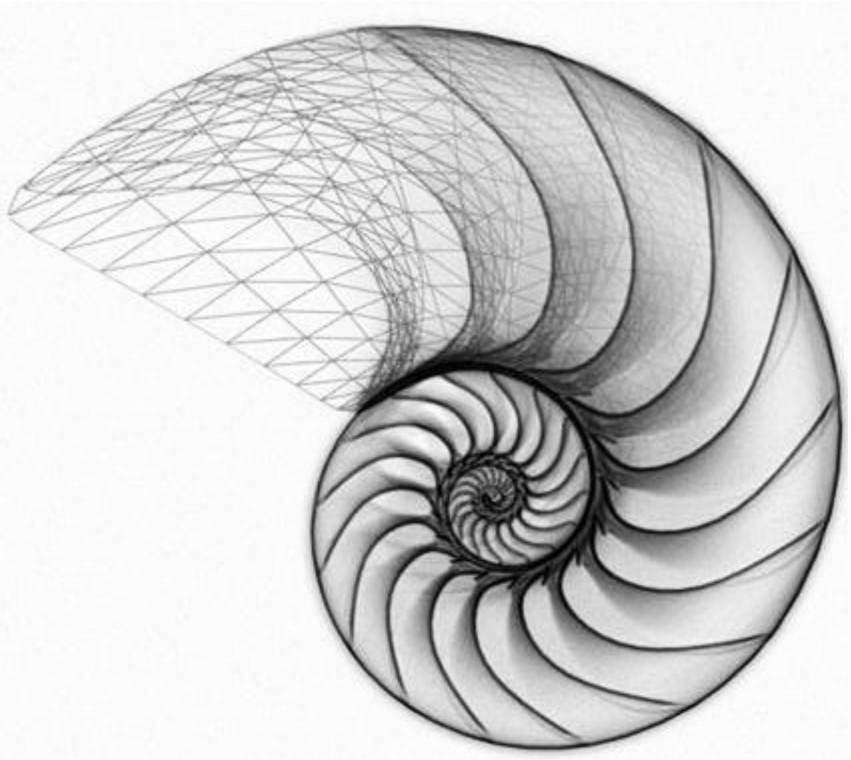
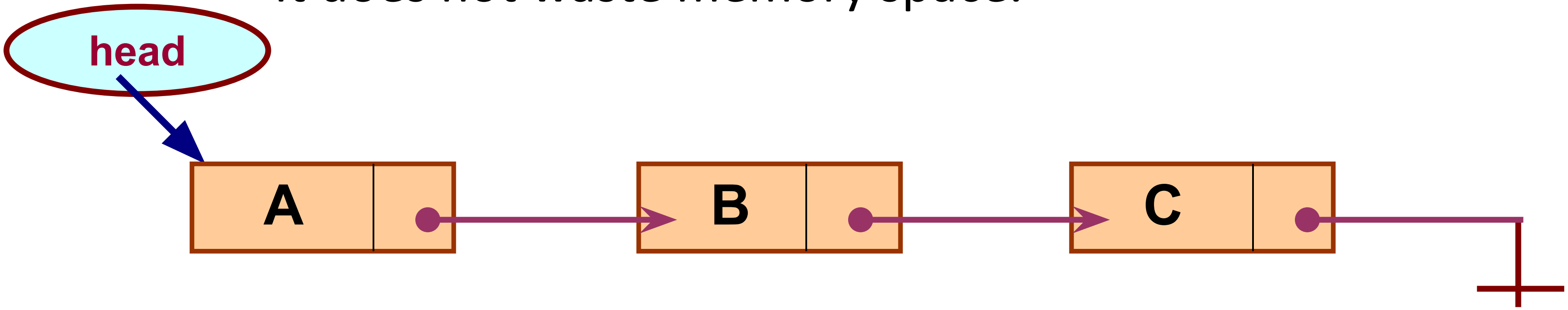


Lecture 12

Abstract Data Type Unsorted List and Sorted List (Linked-list-based Implementation]

Introduction

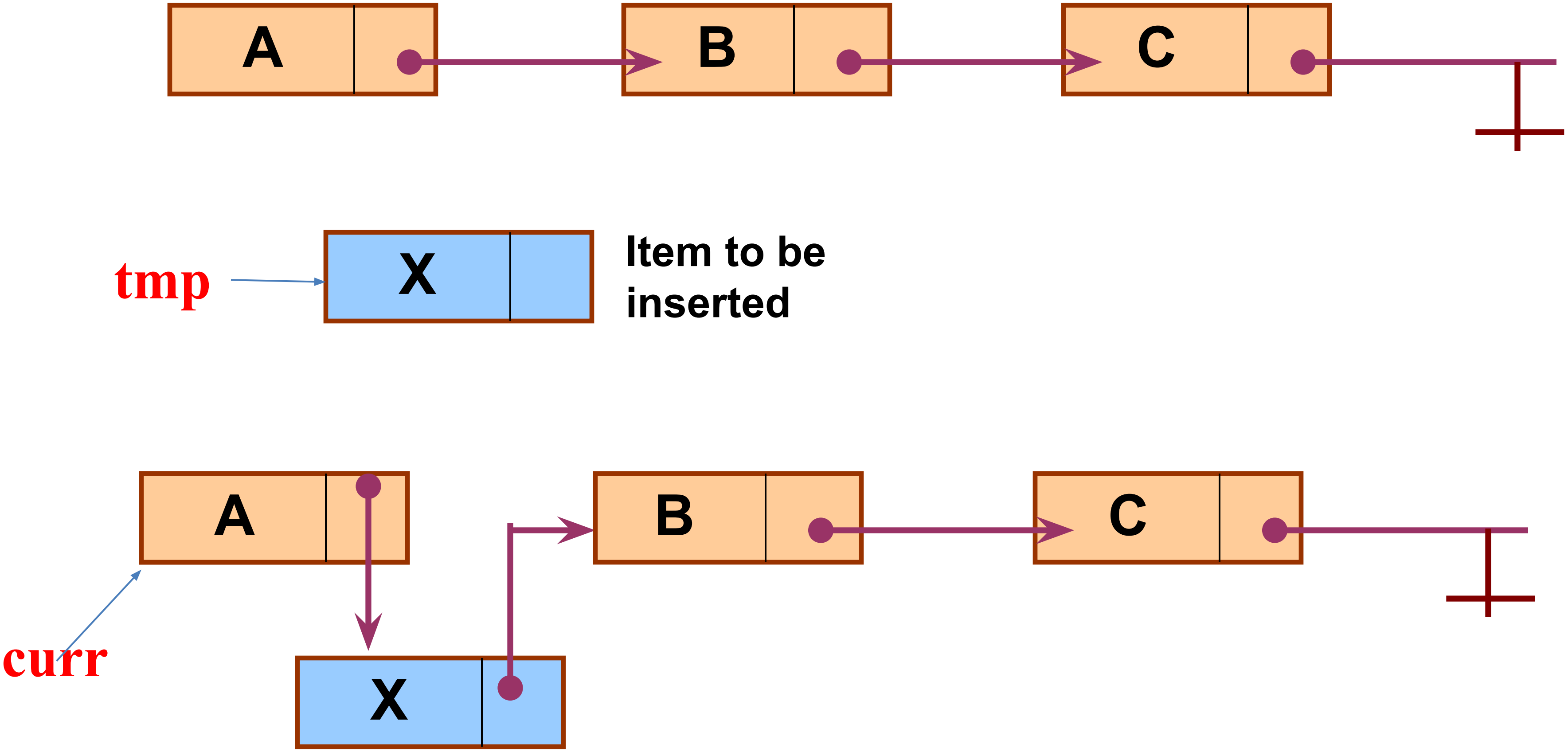
- A linked list is a data structure which can change during execution.
  - Successive elements are connected by pointers.
  - Last element points to NULL.
  - It can grow or shrink in size during execution of a program.
  - It can be made just as long as required.
  - It does not waste memory space.



- Keeping track of a linked list:
  - Must know the pointer to the first element of the list (called *start*, *head*, etc.).

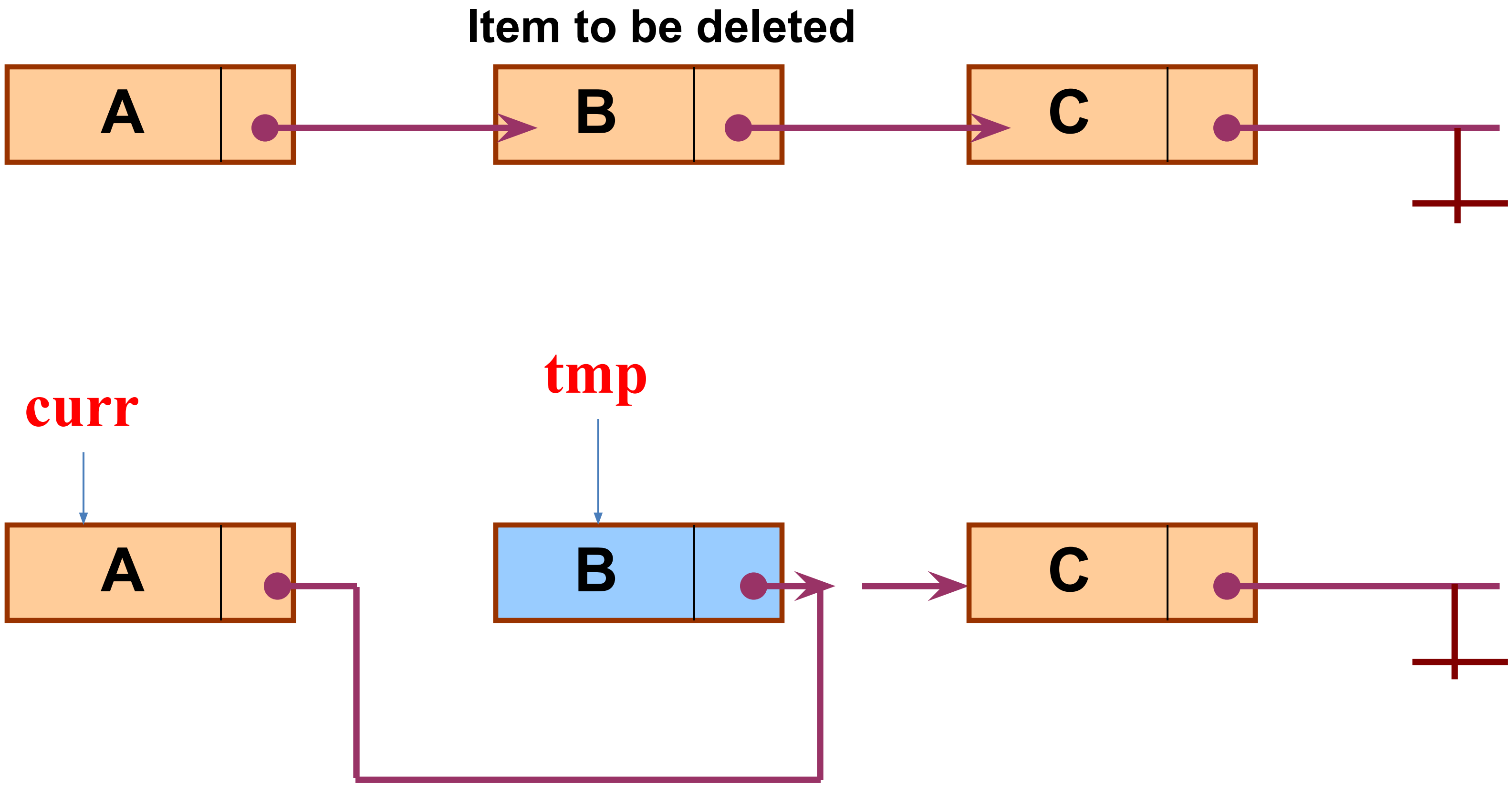
- Linked lists provide flexibility in allowing the items to be rearranged efficiently.
  - Insert an element.
  - Delete an element.

Illustration: Insertion



Item to be inserted

Illustration: Deletion



Array versus Linked Lists

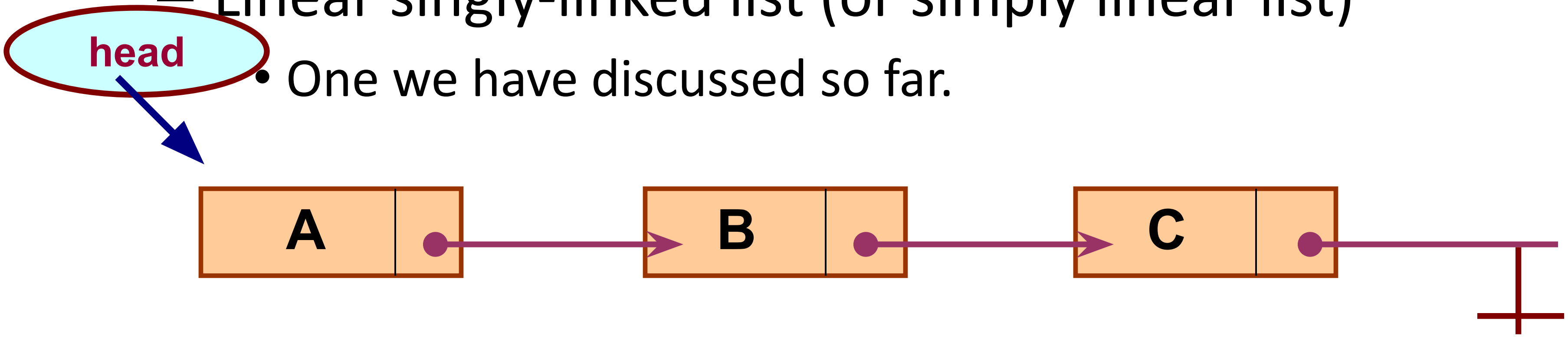
- Arrays are suitable for:
  - Inserting/deleting an element at the end.
  - Randomly accessing any element.
  - Searching the list for a particular value.
- Linked lists are suitable for:
  - Inserting an element.
  - Deleting an element.
  - Applications where sequential access is required.
  - In situations where the number of elements cannot be predicted beforehand.

Types of Lists

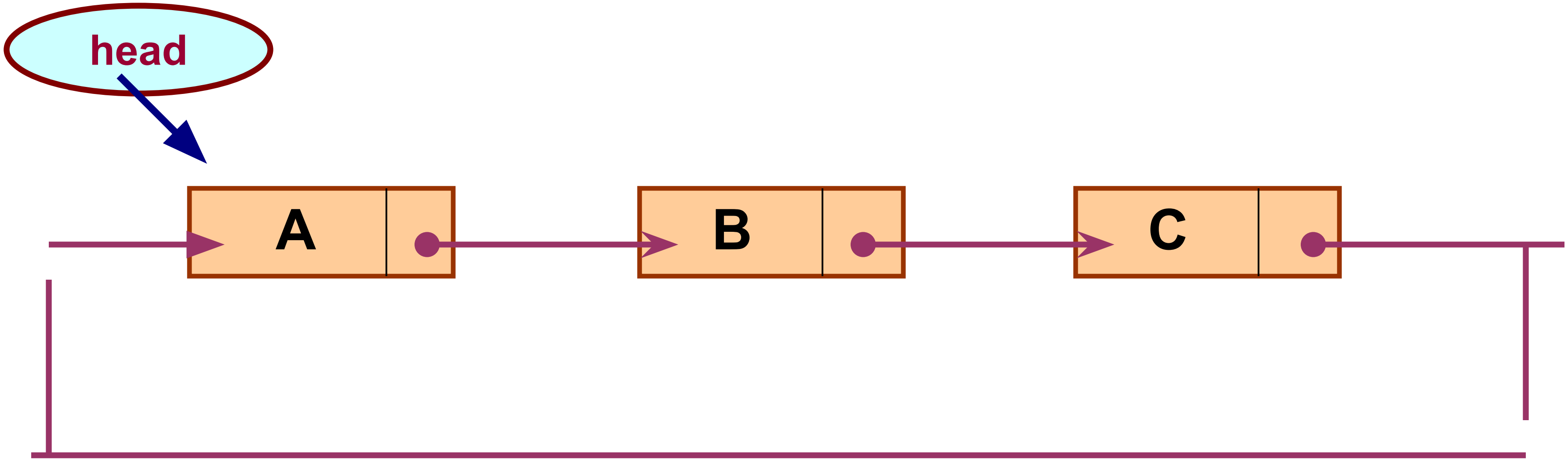
- Depending on the way in which the links are used to maintain adjacency, several different types of linked lists are possible.

  – Linear singly-linked list (or simply linear list)

    - One we have discussed so far.

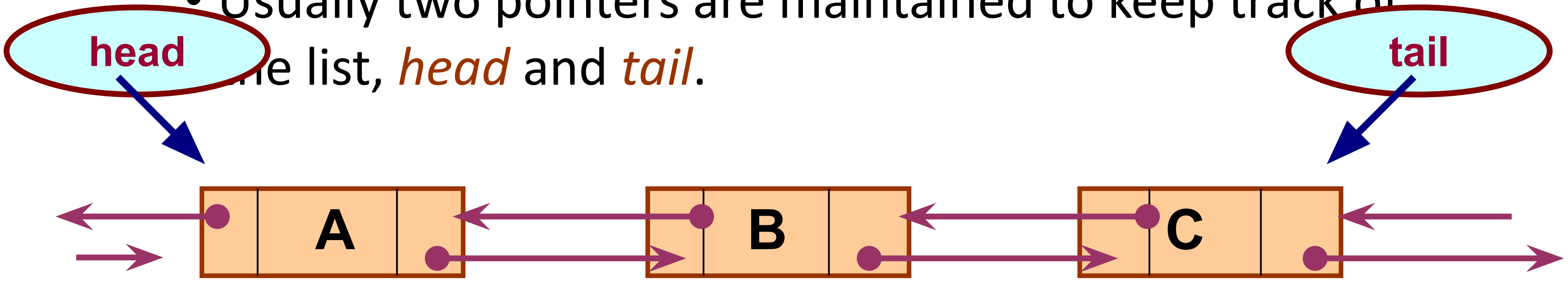


- Circular linked list
  - The pointer from the last element in the list points back to the first element.



– Doubly linked list

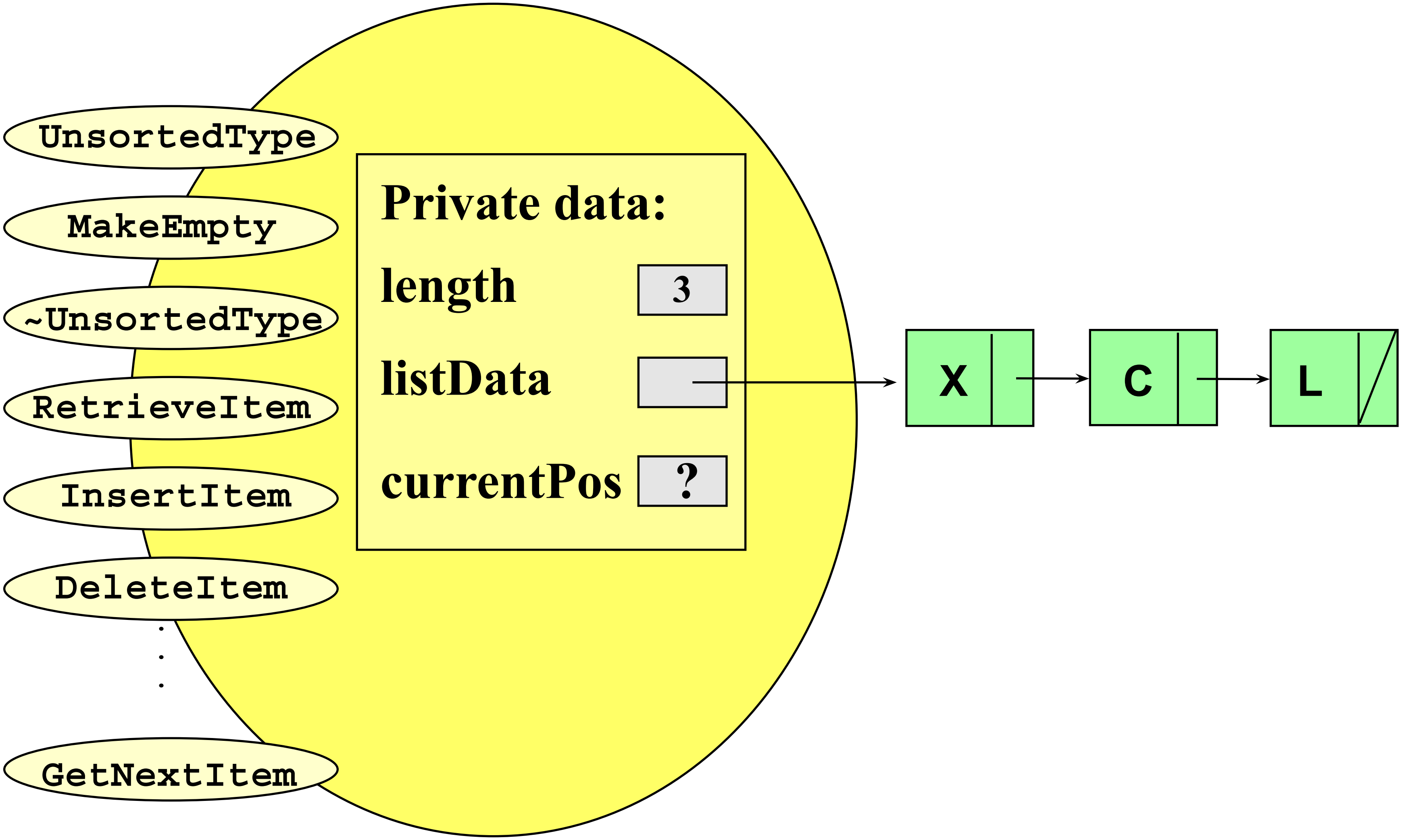
- Pointers exist between adjacent nodes in both directions.
- The list can be traversed either forward or backward.
- Usually two pointers are maintained to keep track of the list, *head* and *tail*.



Basic Operations on a List

- Creating a list
- Traversing the list
- Inserting an item in the list
- Deleting an item from the list
- Concatenating two lists into one

class UnsortedType<char>



UnsortedType

MakeEmpty

~UnsortedType

RetrieveItem

InsertItem

DeleteItem

. . .

GetNextItem

Private data:

length 3

listData

currentPos ?

unsortedlinkedlist.h

```
#ifndef UNSORTEDLINKEDLIST_H_INCLUDED
#define UNSORTEDLINKEDLIST_H_INCLUDED

template <class ItemType>
class UnsortedType
{
  struct NodeType
  {
    ItemType info;
    NodeType* next;
  };

public:
  UnsortedType();
  ~UnsortedType();
  bool IsFull();
  int LengthIs();
  void MakeEmpty();
  void RetrieveItem(ItemType& item, bool& found);
  void InsertItem(ItemType item);
  void DeleteItem(ItemType item);
  void ResetList();
  void GetNextItem(ItemType& item);


private:
  NodeType* listData;
  int length;
  NodeType* currentPos;
};

#endif // UNSORTEDLINKEDLIST_H_INCLUDED
```

unsortedlinkedlist.cpp

```
#include "unsortedlinkedlist.h"
#include<cstddef>
#include<new>

template <class ItemType>
UnsortedType<ItemType>::UnsortedType()
{
  length = 0;
  listData = NULL;
  currentPos = NULL;
}

template <class ItemType>
int UnsortedType<ItemType>::LengthIs()
{
  return length;
}
```

```
template<class ItemType>
bool UnsortedType<ItemType>::IsFull()
{
  NodeType* location;
  try
  {
    location = new NodeType;
    delete location;
    return false;
  }
  catch(std::bad_alloc& exception)
  {
    return true;
  }
}
```

unsortedlinkedlist.cpp

```
#include "unsortedlinkedlist.h"
#include<cstddef>
#include<new>

template <class ItemType>
UnsortedType<ItemType>::UnsortedType()
{
  length = 0;
  listData = NULL;
  currentPos = NULL;
}
```

O(1)

```
template <class ItemType>
int UnsortedType<ItemType>::LengthIs()
{
  return length;
}
```

O(1)

```
template<class ItemType>
bool UnsortedType<ItemType>::IsFull()
{
  NodeType* location;
  try
  {
    location = new NodeType;
    delete location;
    return false;
  }
  catch(std::bad_alloc& exception)
  {
    return true;
  }
}
```

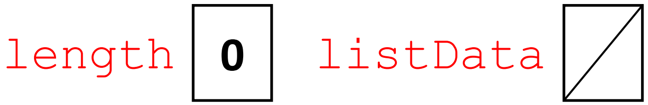
O(1)

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```

unsortedlinkedlist.cpp

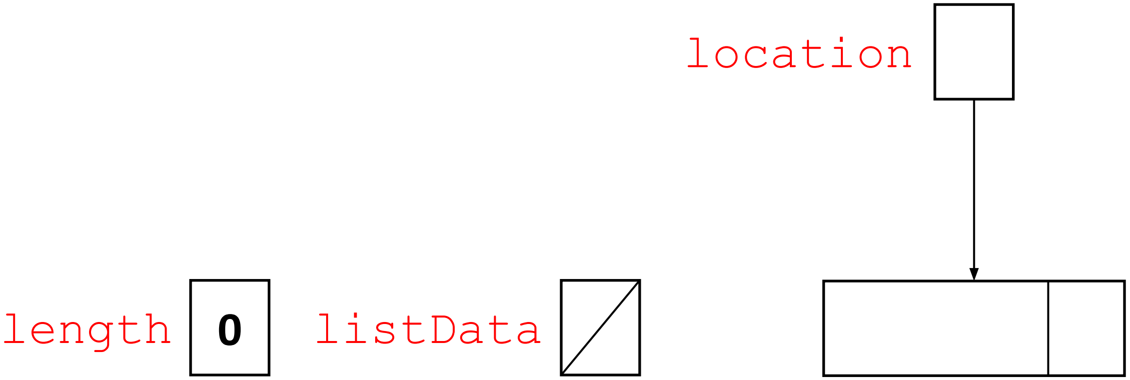
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('X')

unsortedlinkedlist.cpp

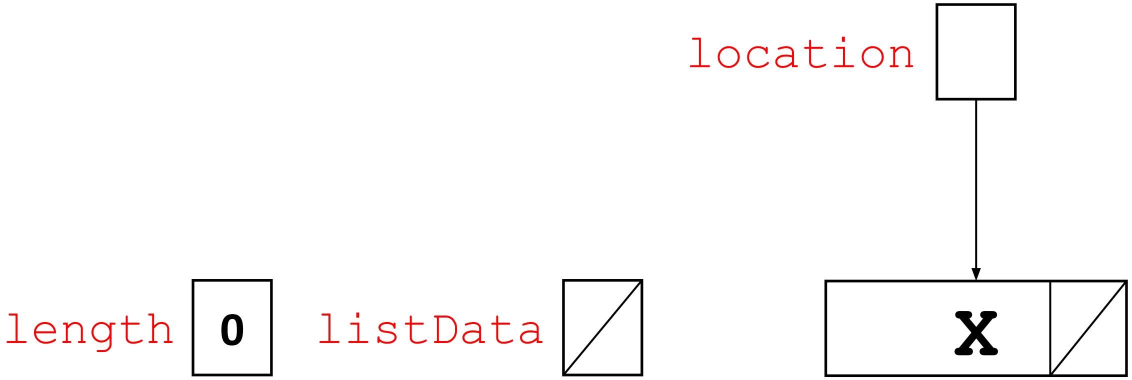
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('X')

unsortedlinkedlist.cpp

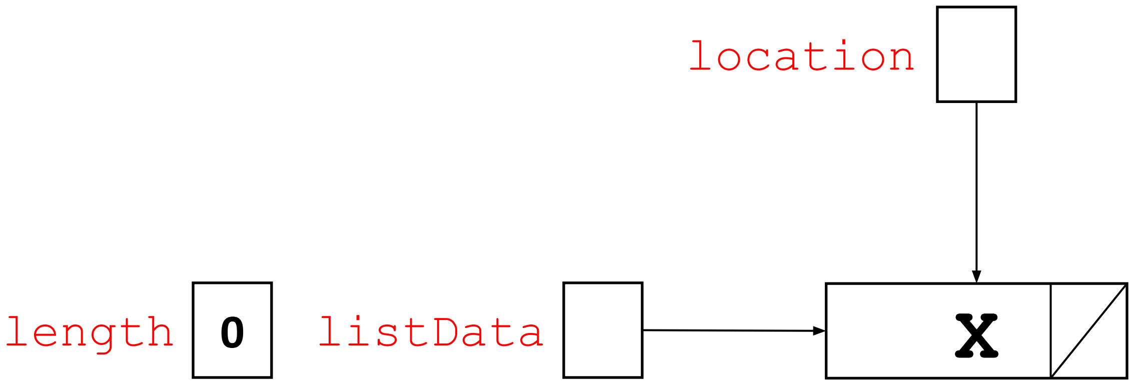
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('X')

unsortedlinkedlist.cpp

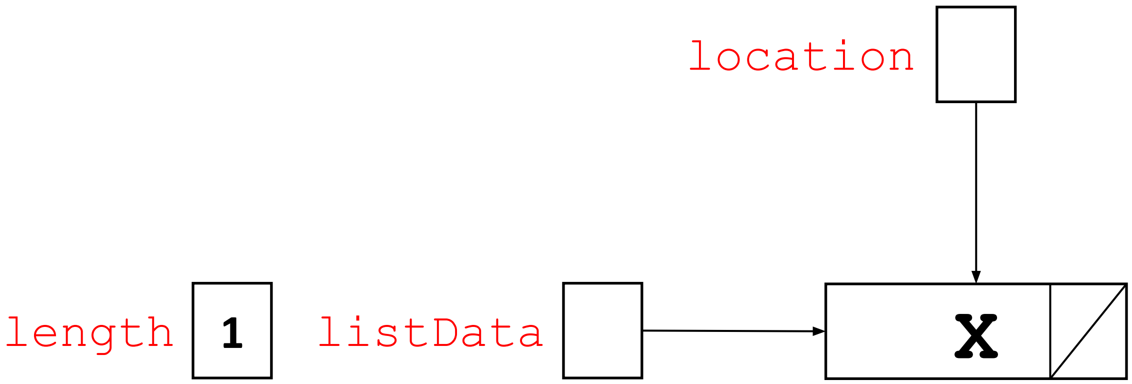
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('X')

unsortedlinkedlist.cpp

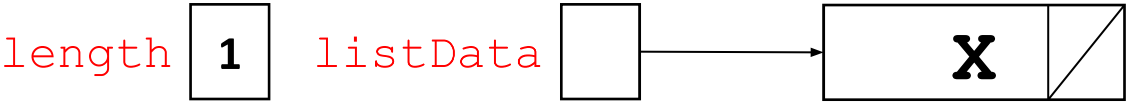
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('X')

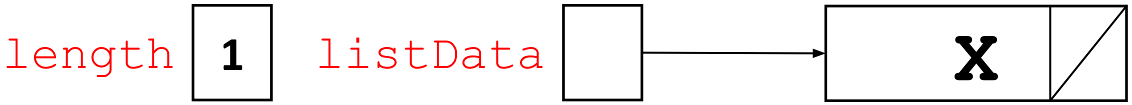
unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



unsortedlinkedlist.cpp

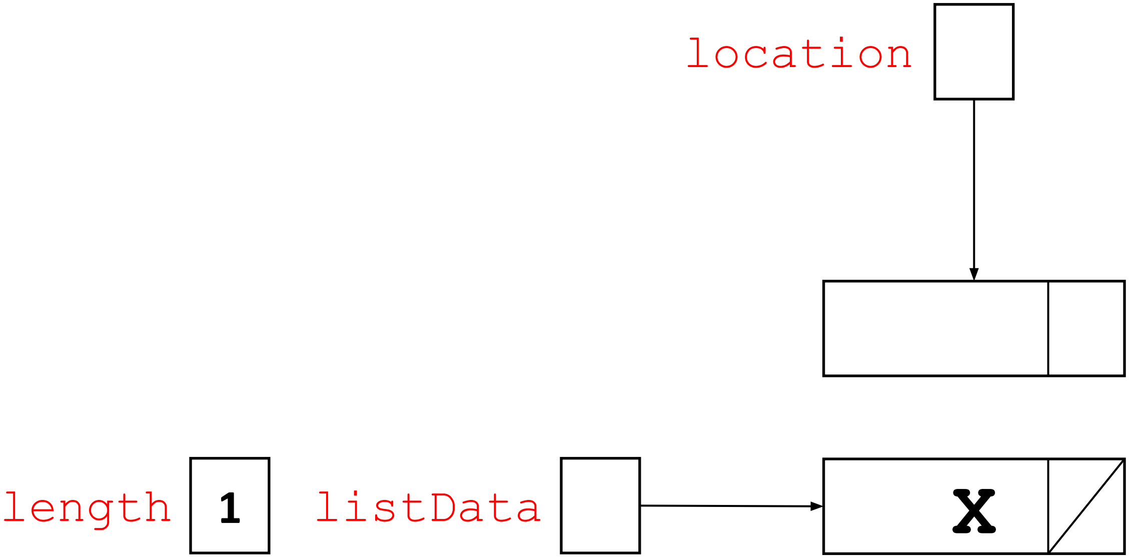
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('C')

unsortedlinkedlist.cpp

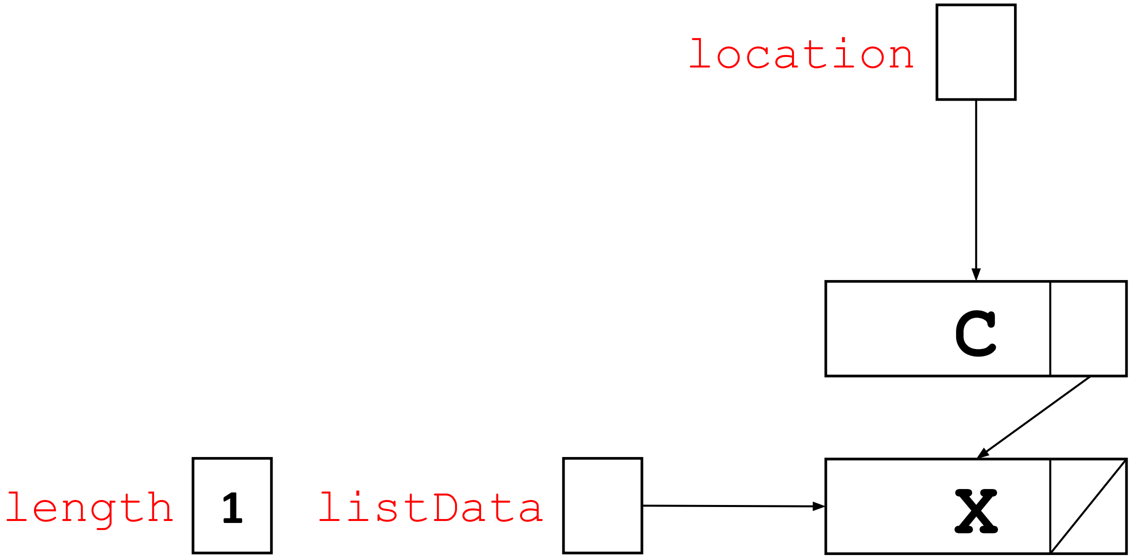
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('C')

unsortedlinkedlist.cpp

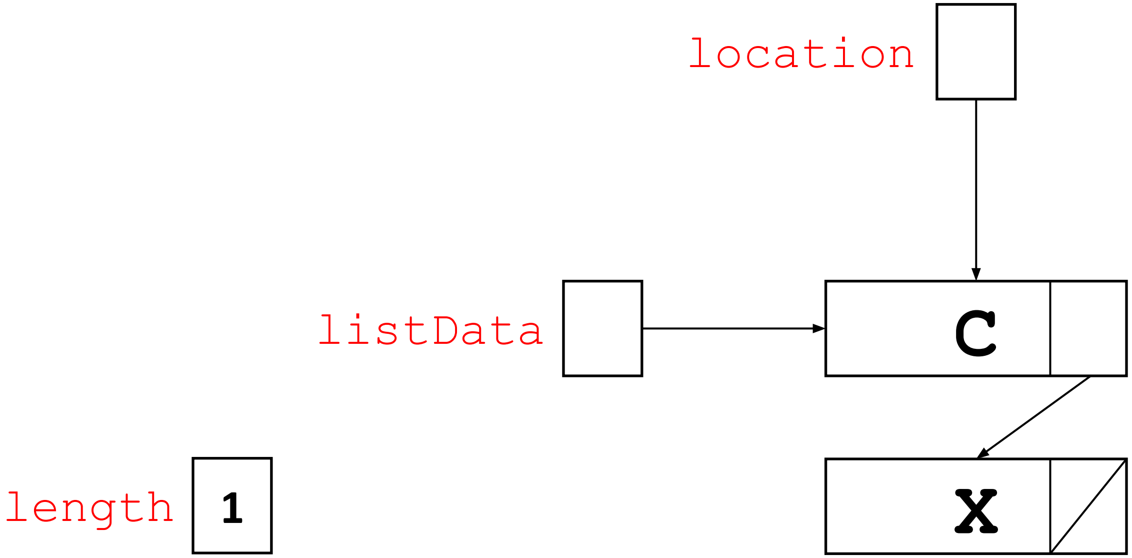
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('C')

unsortedlinkedlist.cpp

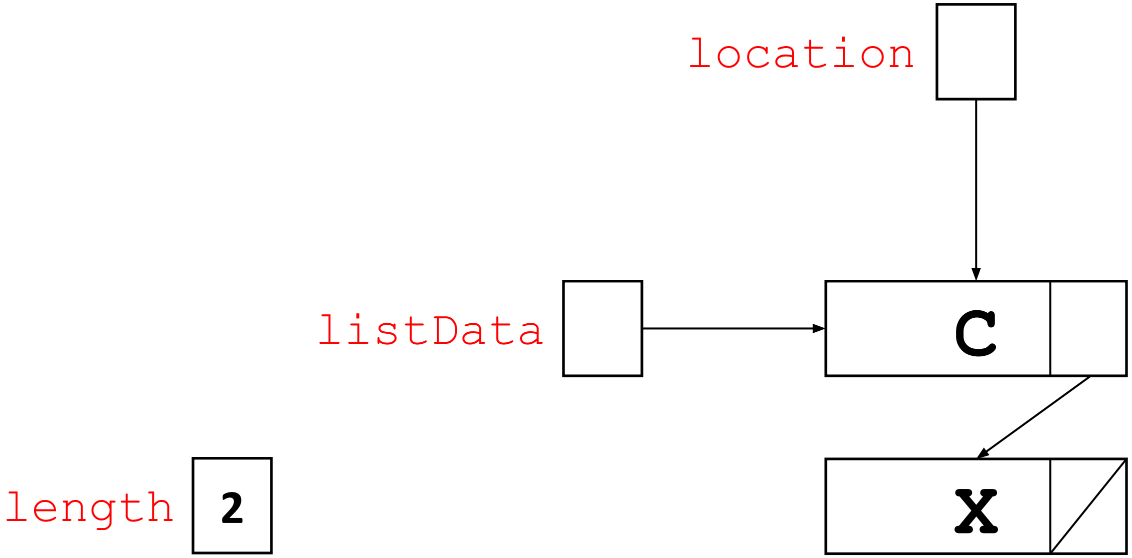
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('C')

unsortedlinkedlist.cpp

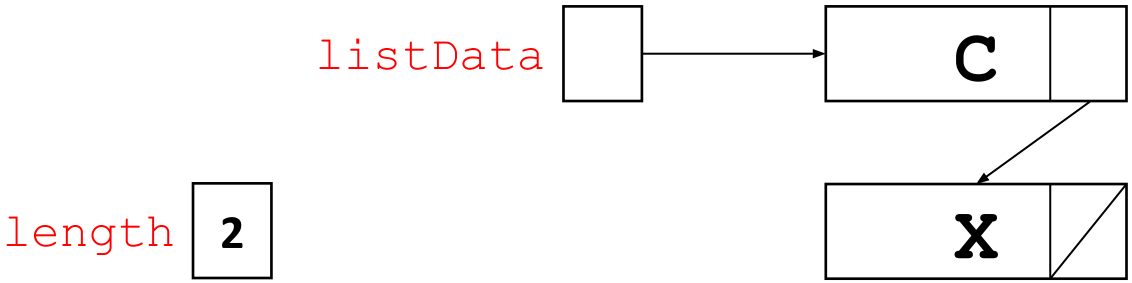
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('C')

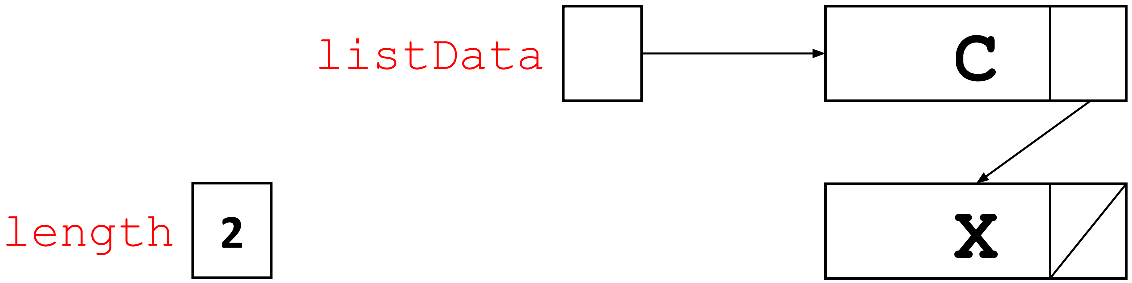
unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



unsortedlinkedlist.cpp

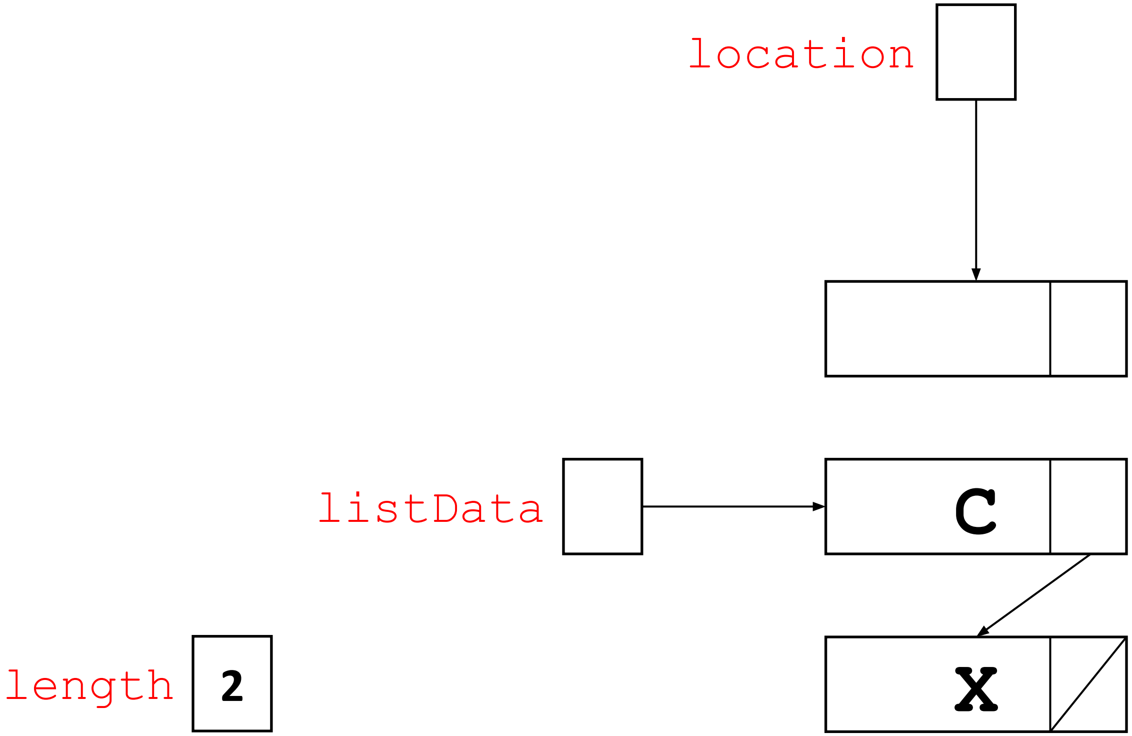
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem(‘L’)

unsortedlinkedlist.cpp

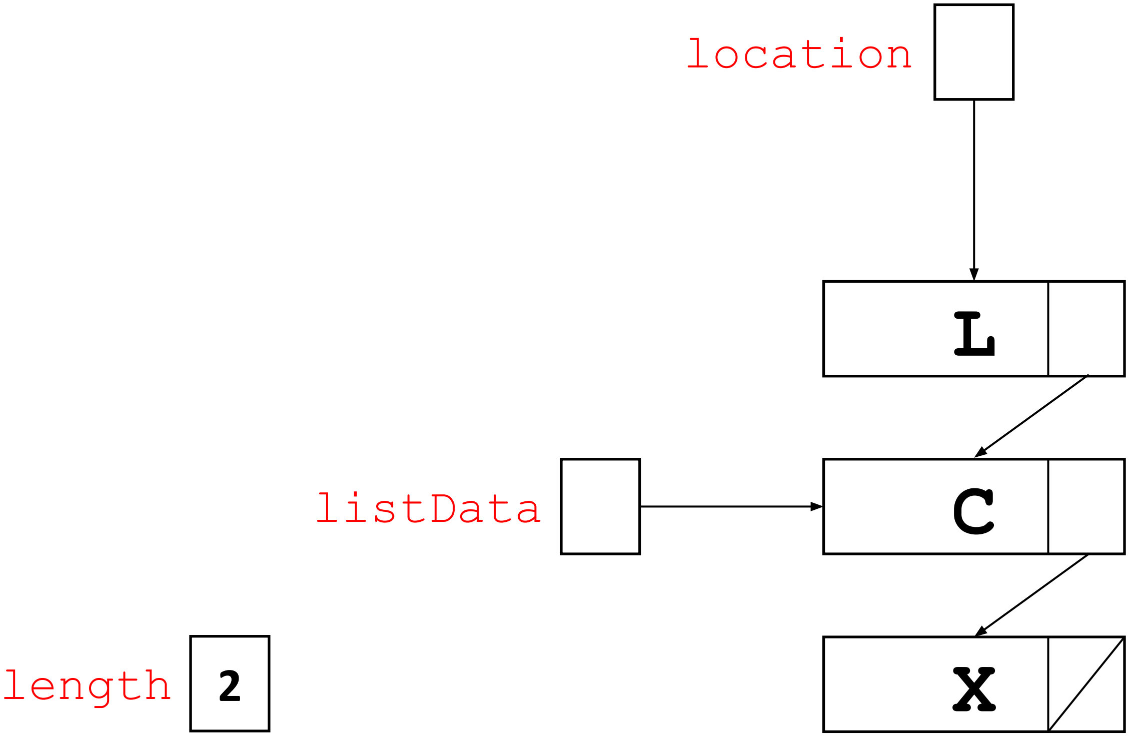
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('L')

unsortedlinkedlist.cpp

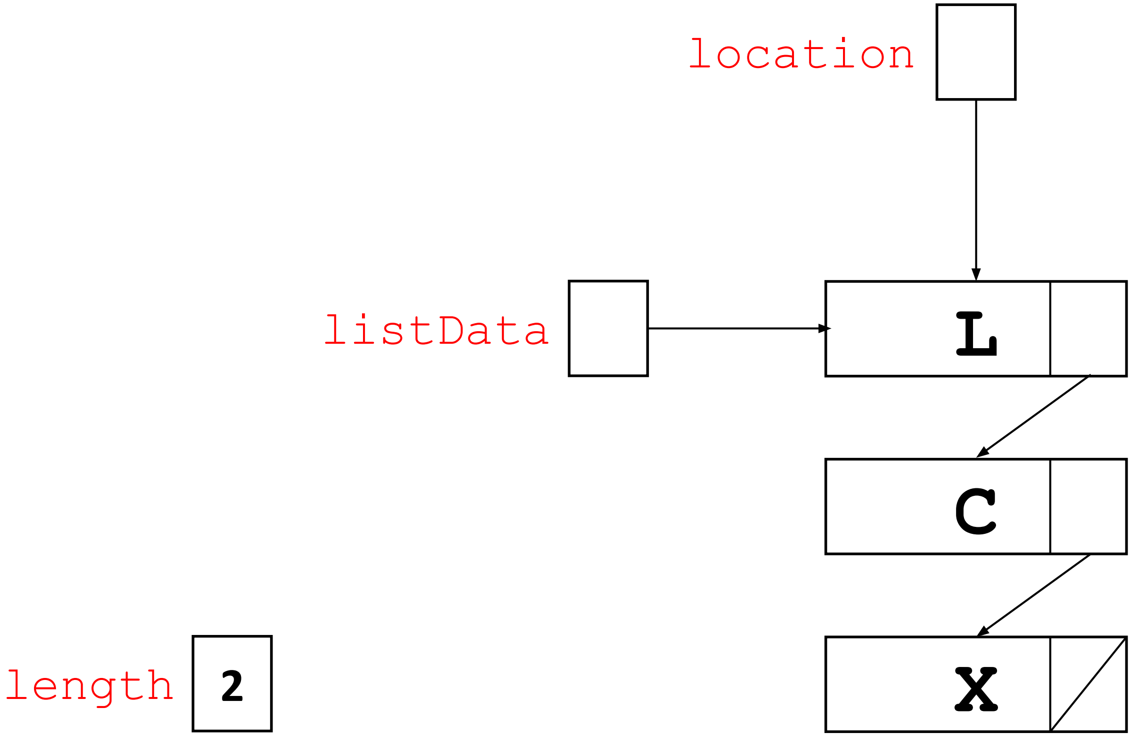
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('L')

unsortedlinkedlist.cpp

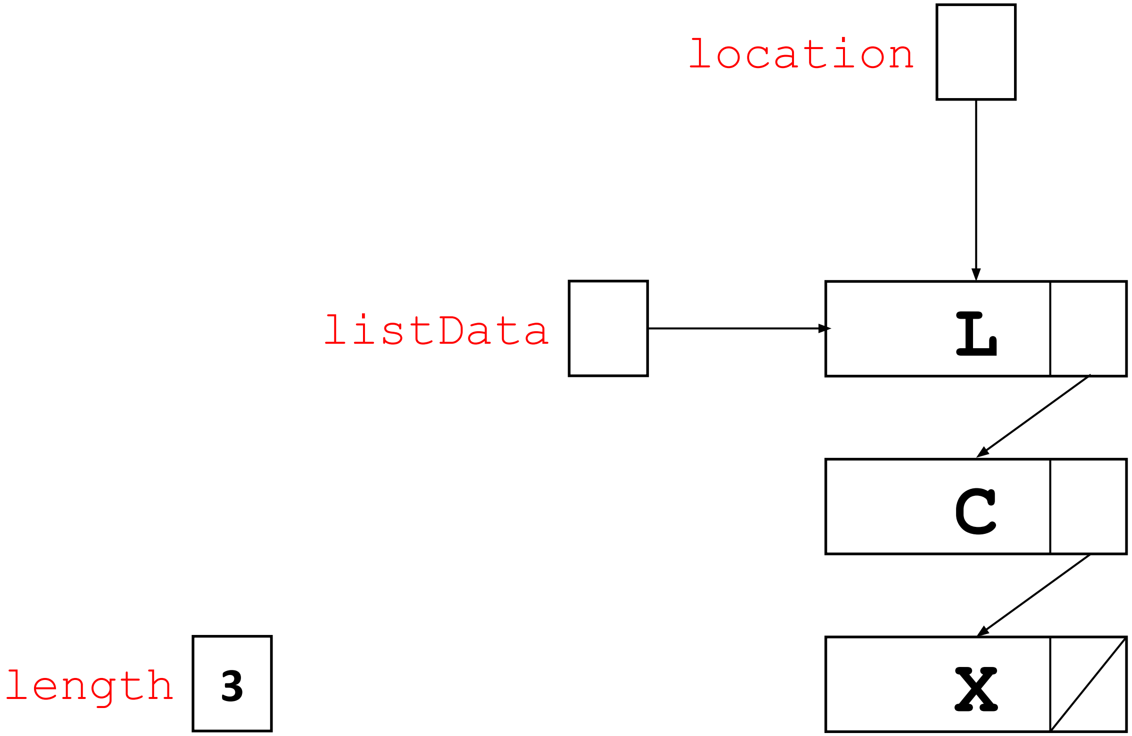
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('L')

unsortedlinkedlist.cpp

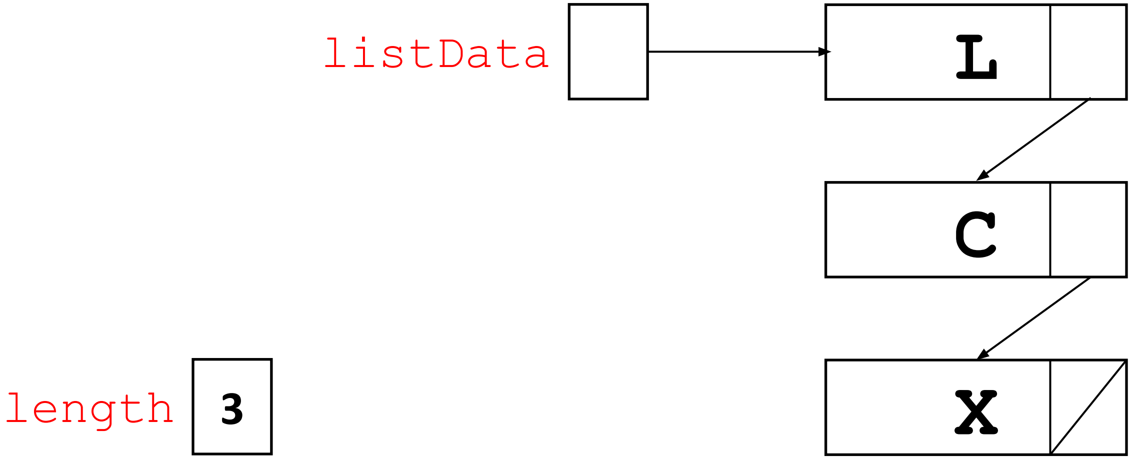
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



InsertItem('L')

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```



unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* location;
  location = new NodeType;
  location->info = item;
  location->next = listData;
  listData = location;
  length++;
}
```

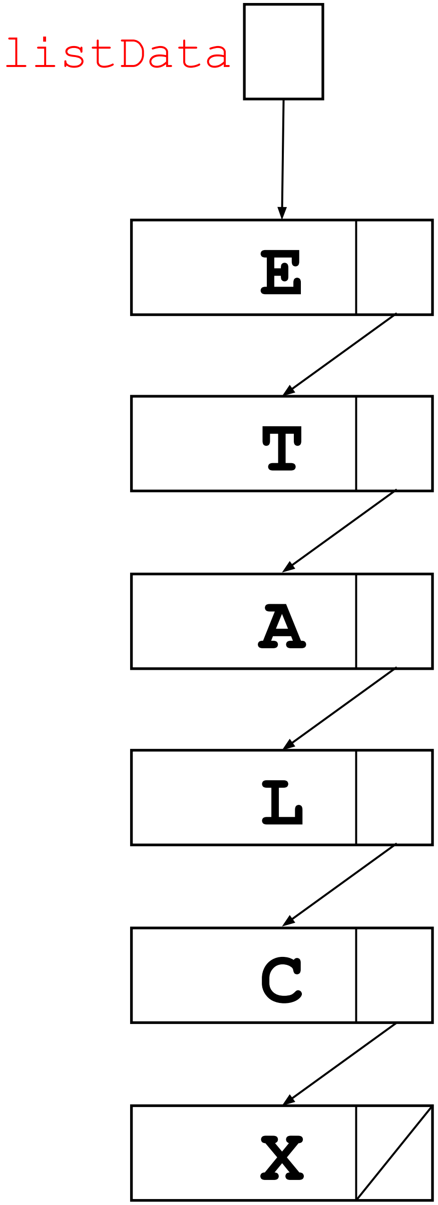
O(1)

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```

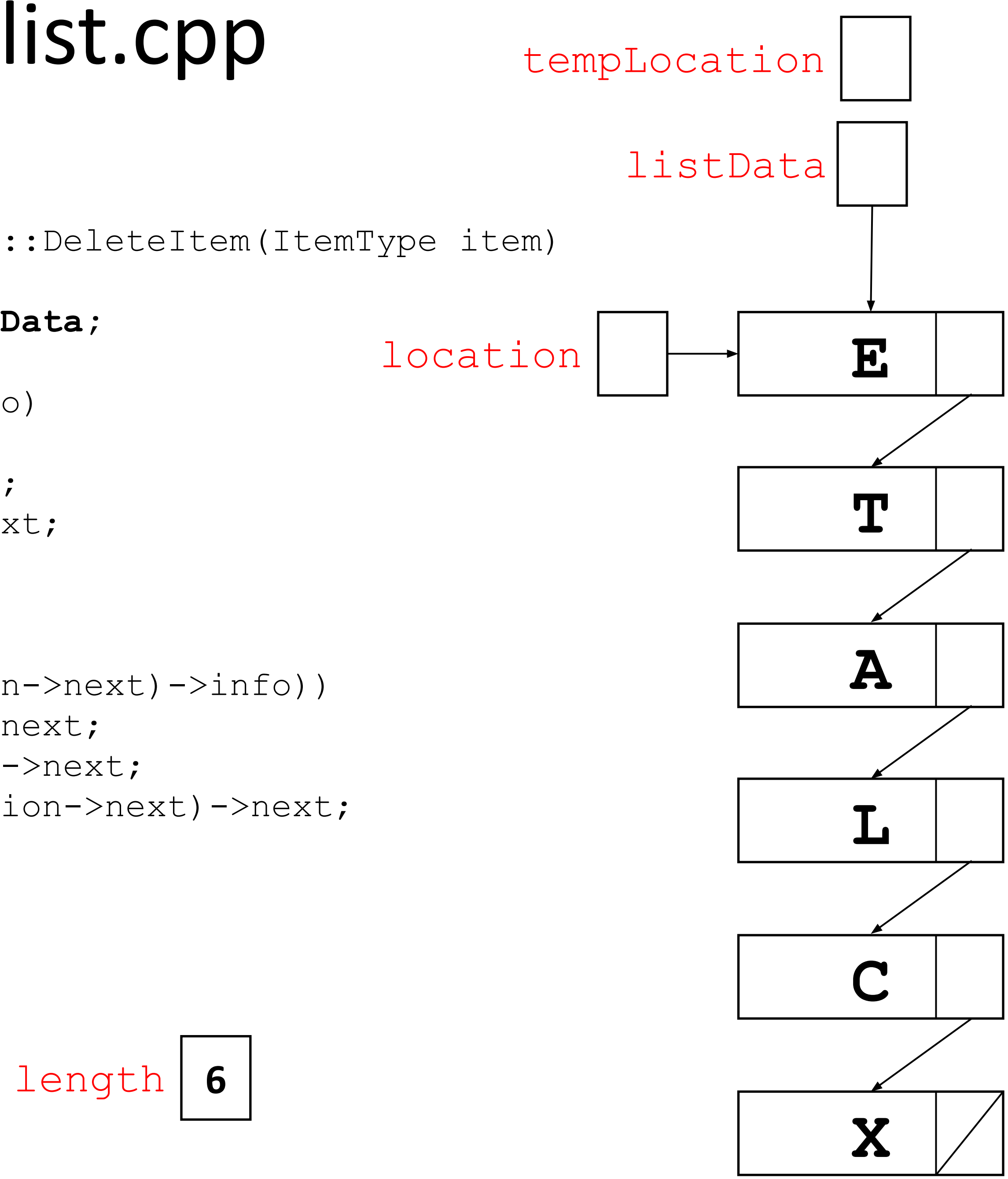


length 6

DeleteItem('L')

unsortedlinkedlist.cpp

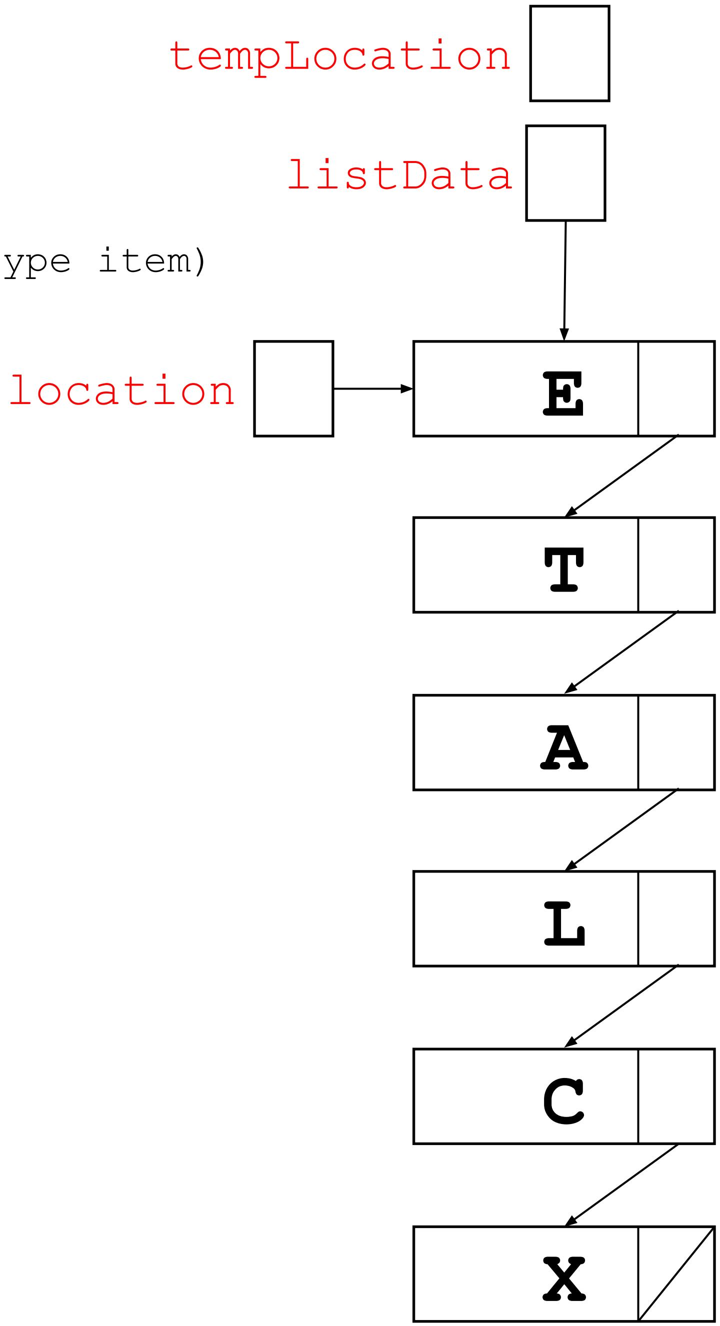
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem('L')

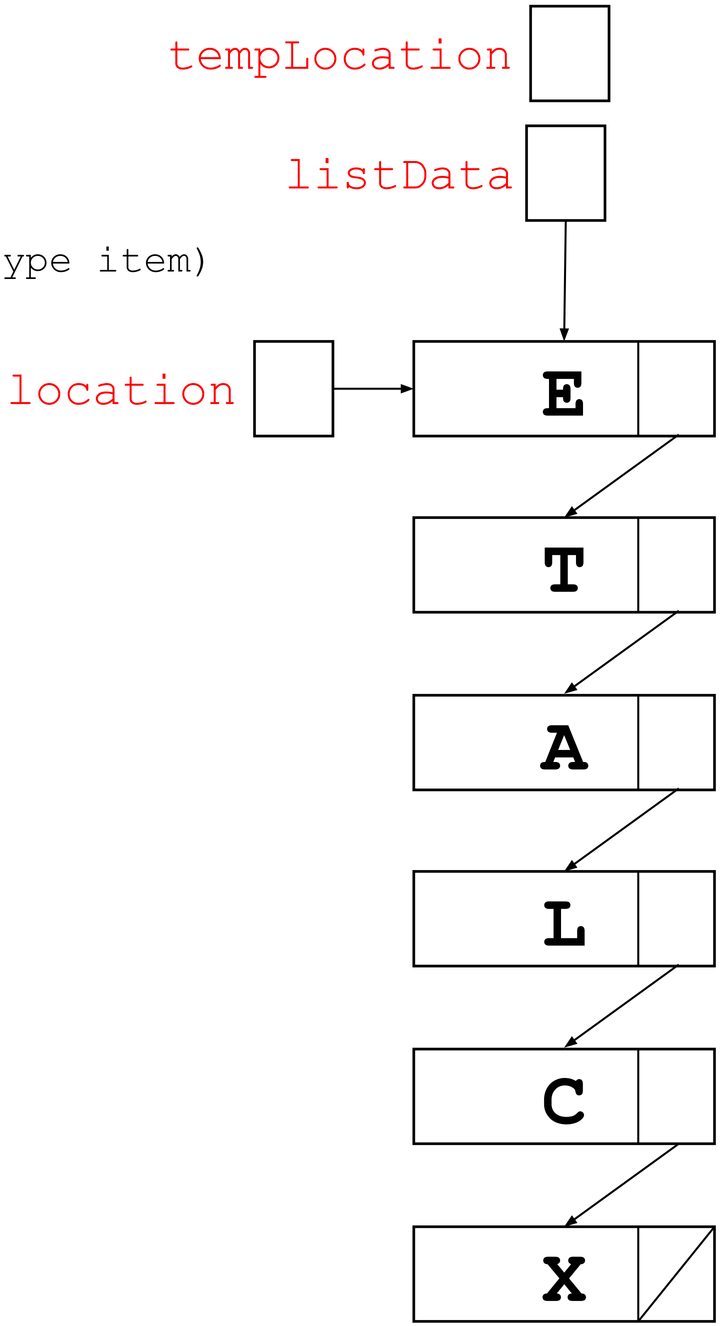
unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem('L')

unsortedlinkedlist.cpp

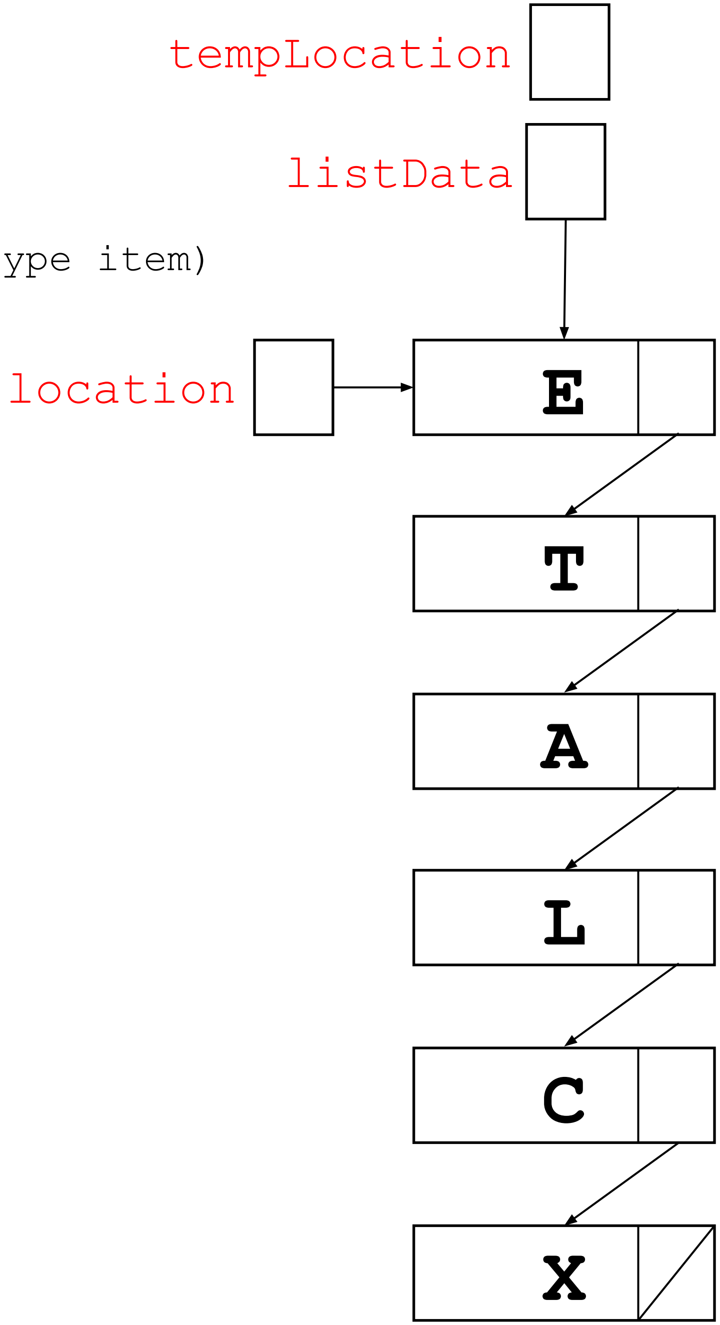


```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```

DeleteItem('L')

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```

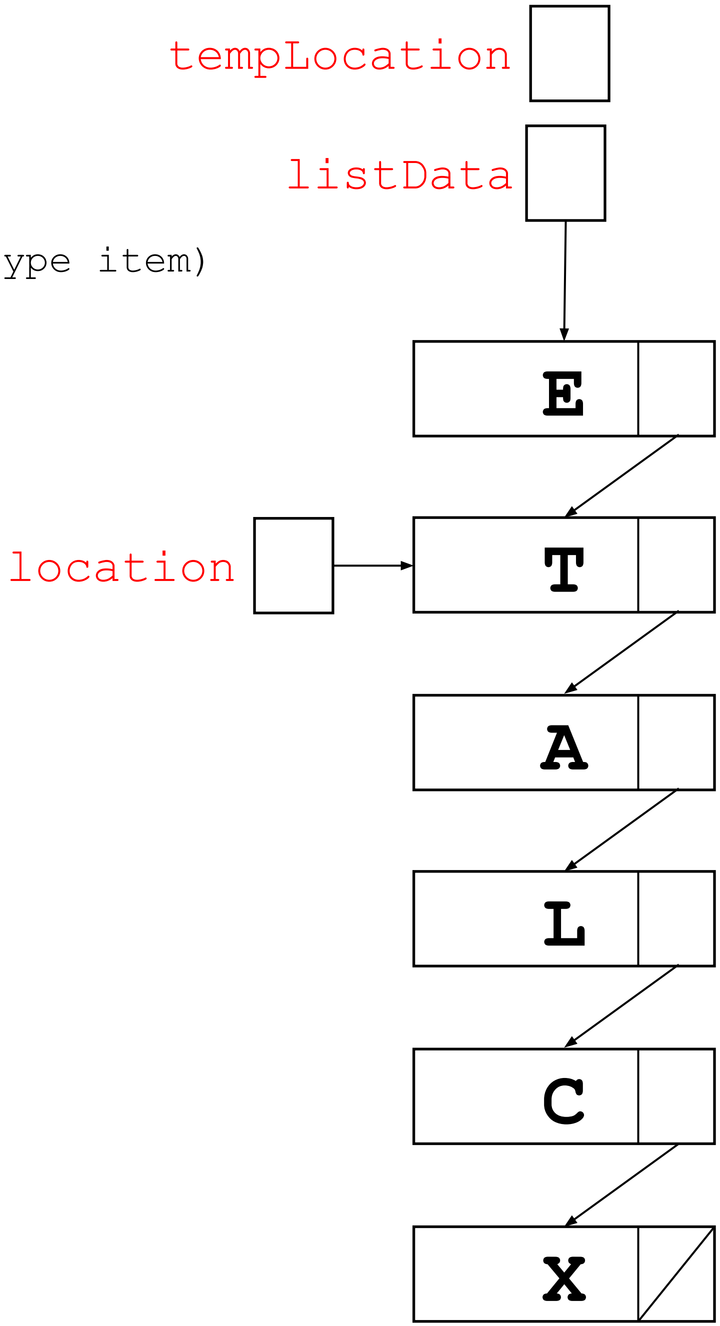


length 6

DeleteItem('L')

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```

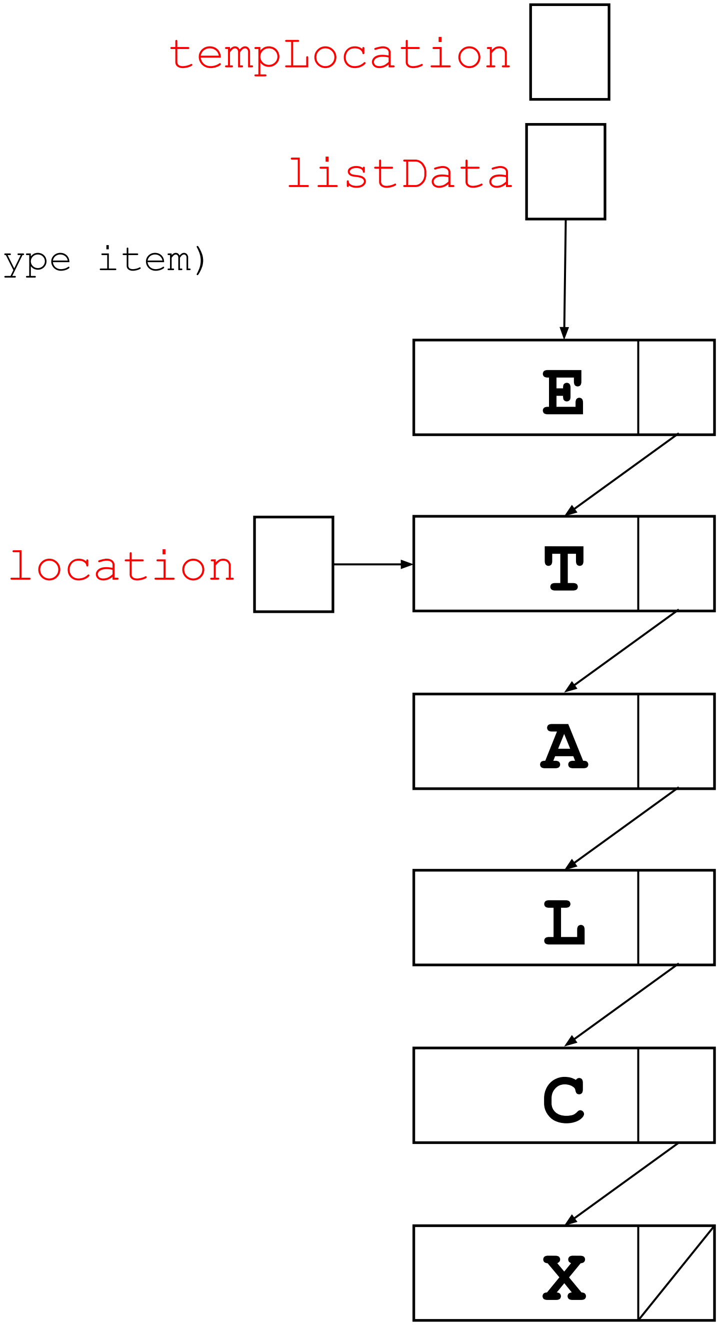


length 6

DeleteItem('L')

unsortedlinkedlist.cpp

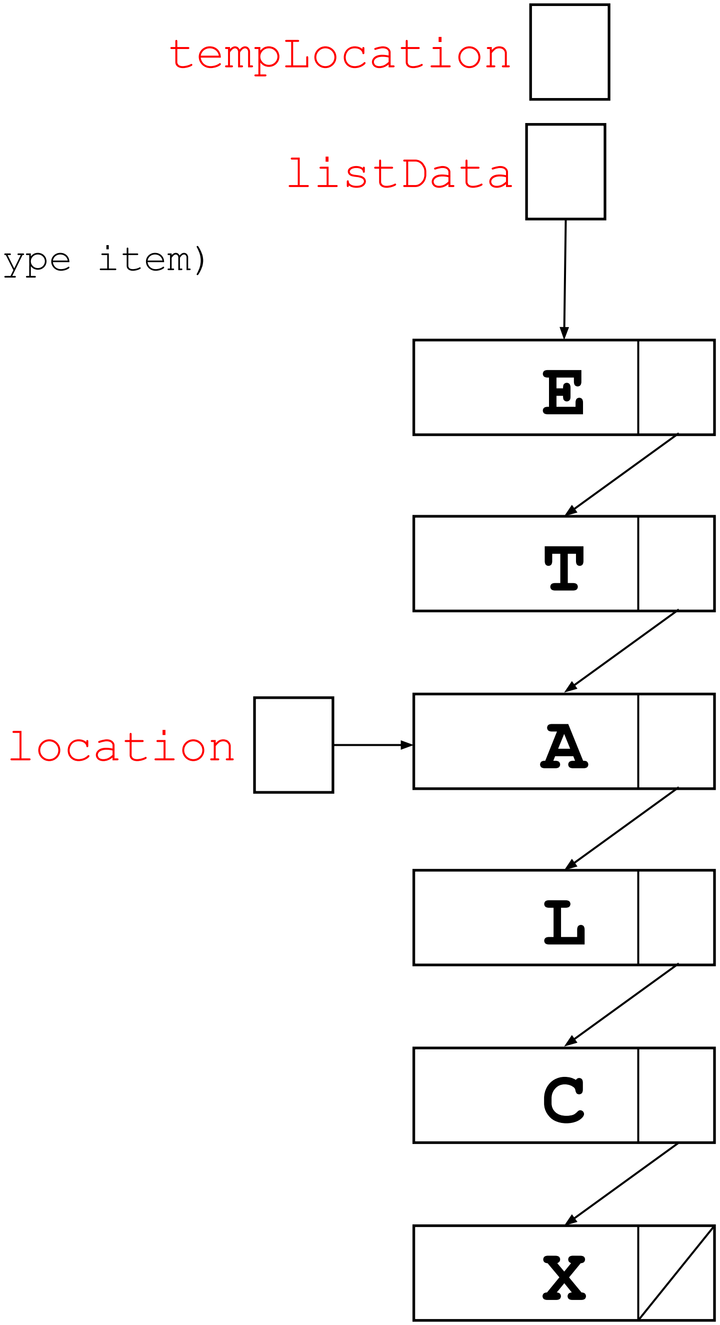
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem('L')

unsortedlinkedlist.cpp

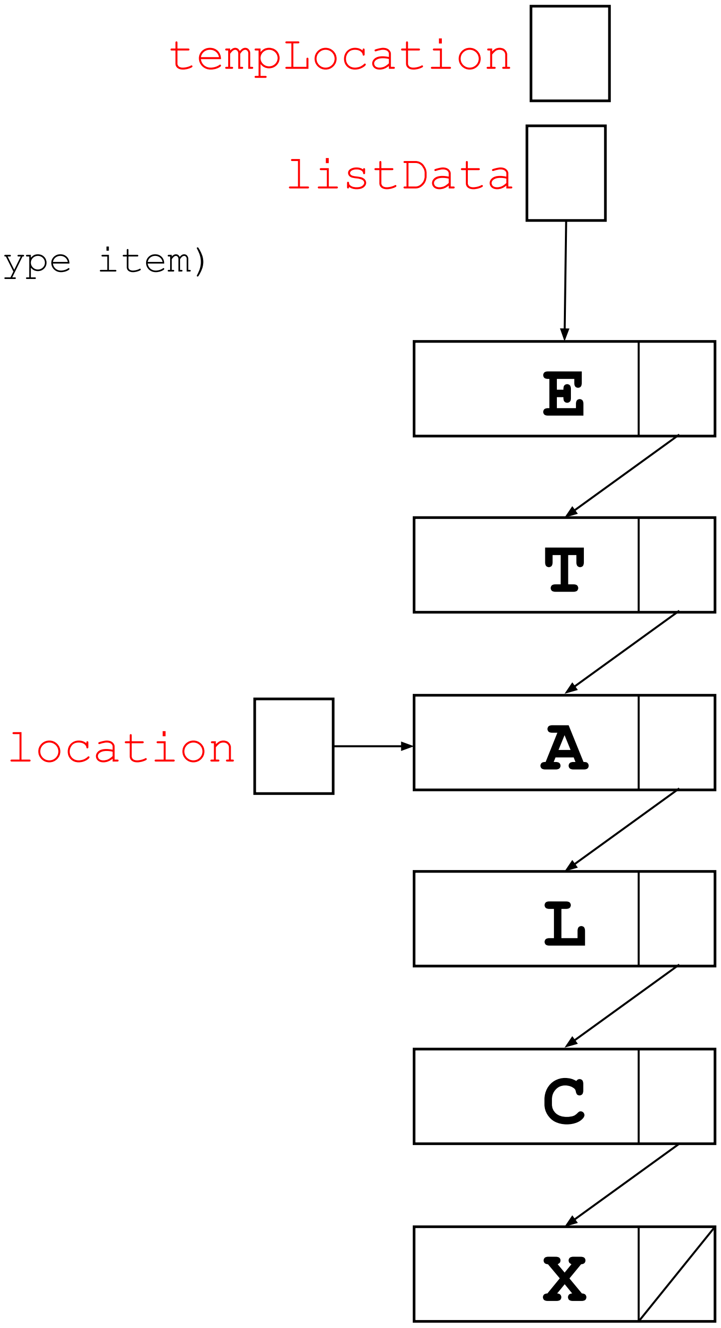
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem('L')

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```

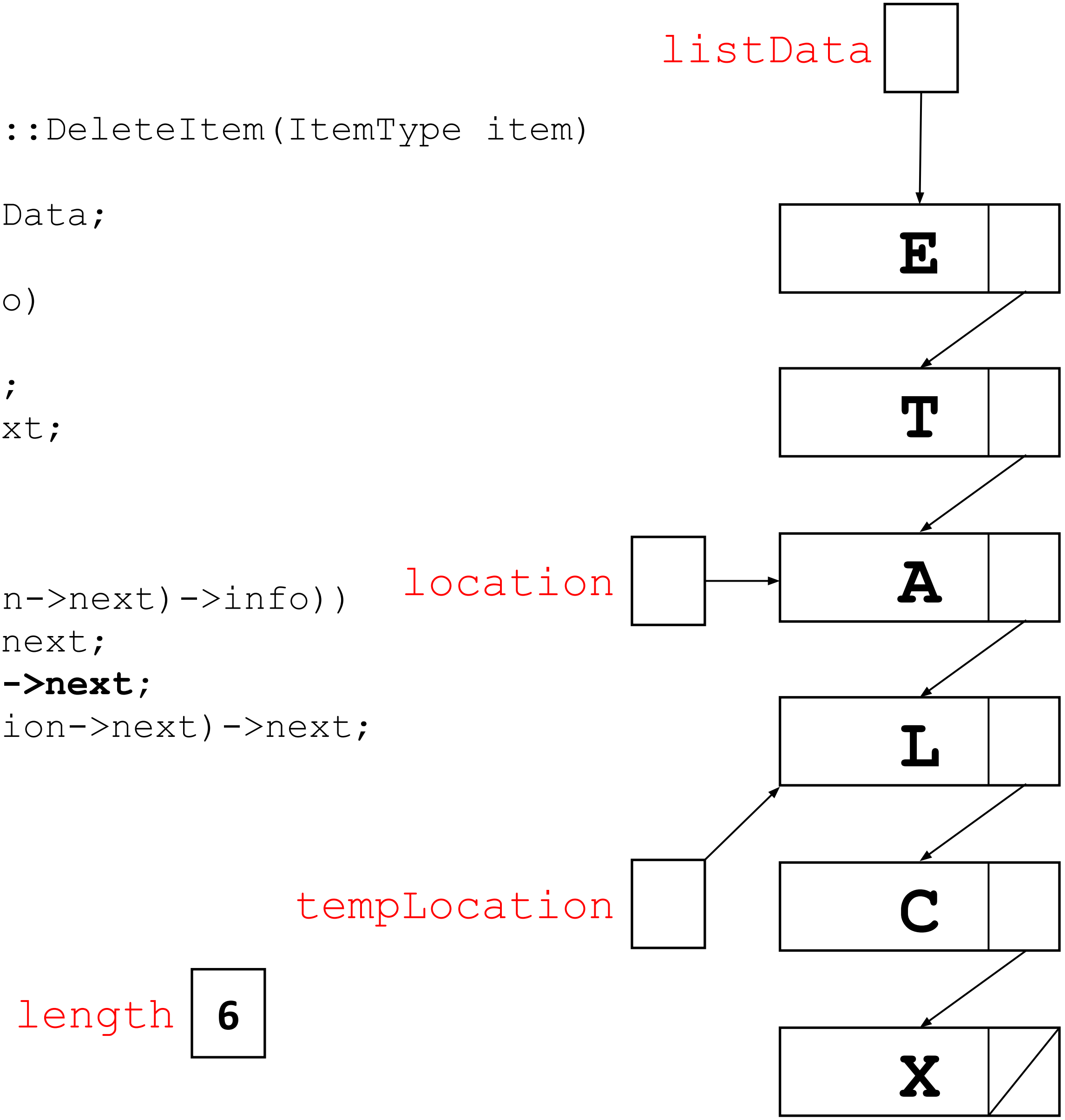


length 6

DeleteItem('L')

unsortedlinkedlist.cpp

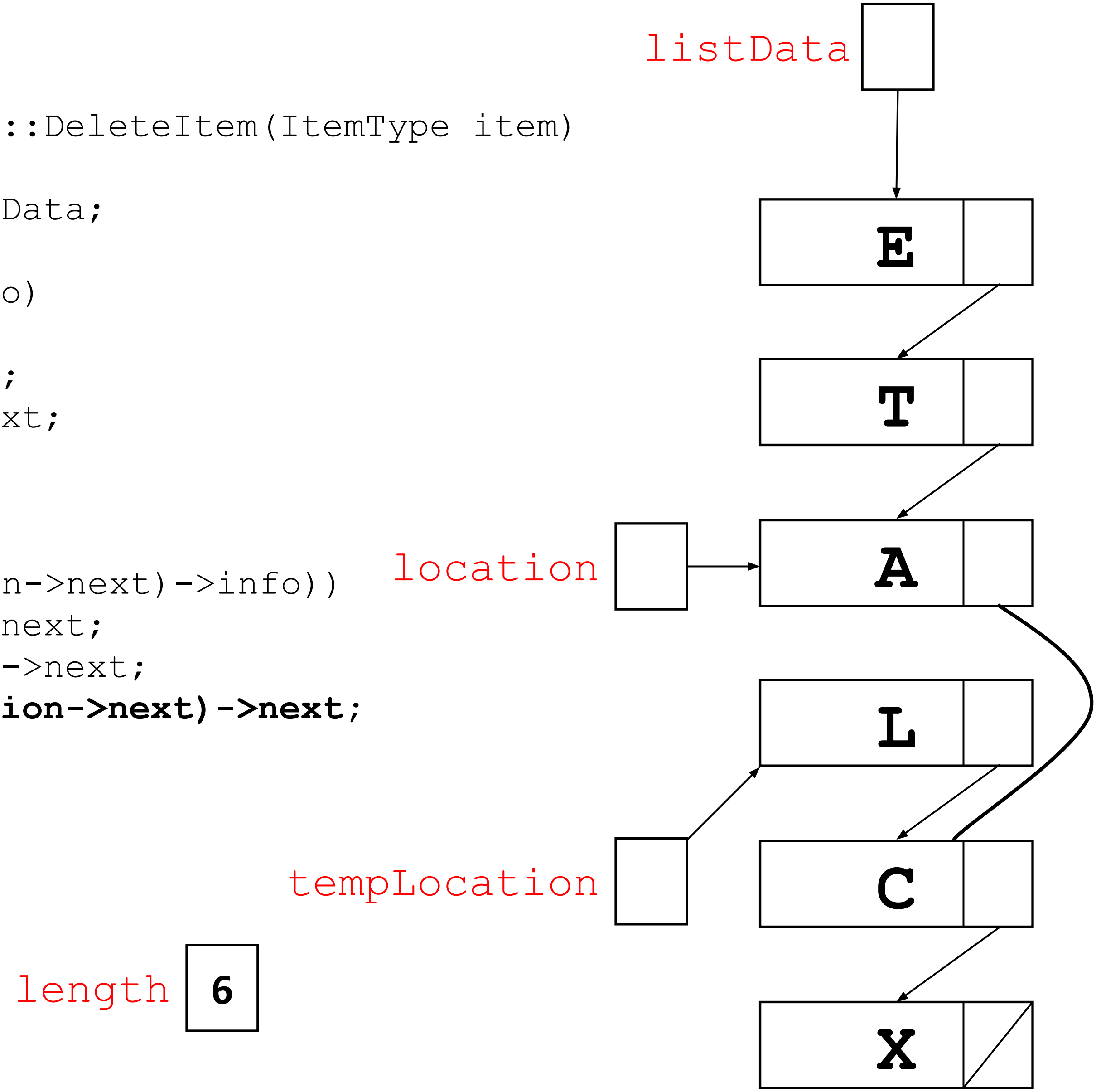
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem('L')

unsortedlinkedlist.cpp

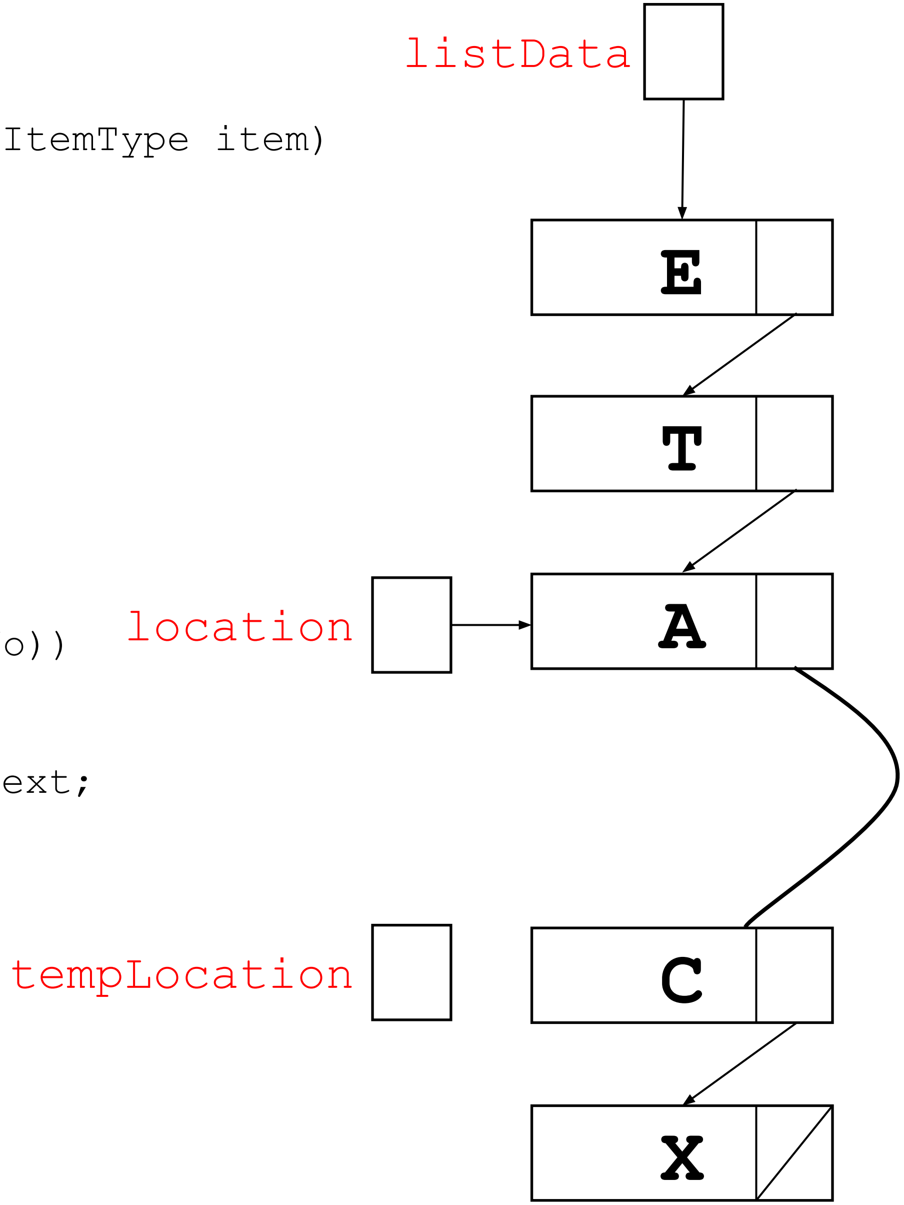
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem(‘L’)

unsortedlinkedlist.cpp

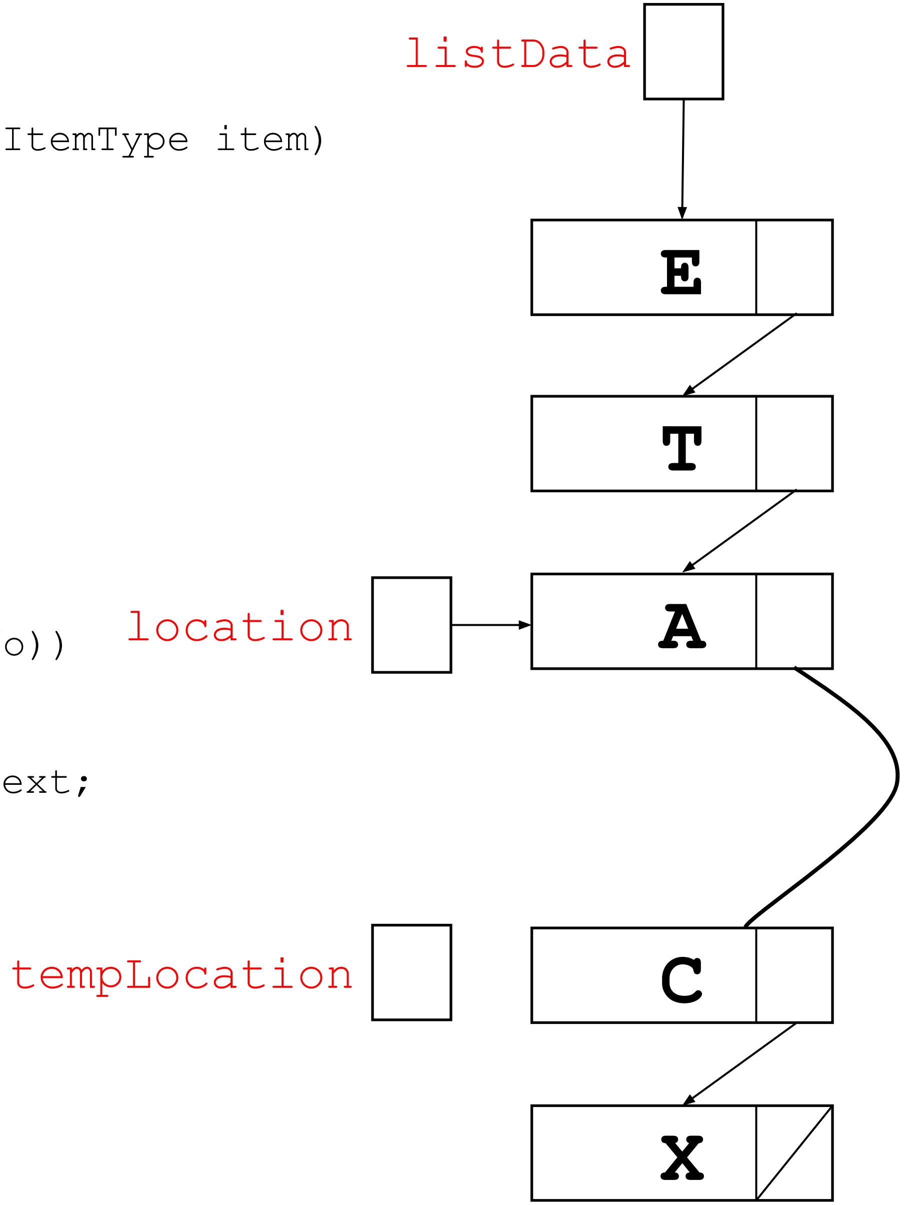
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem('L')

unsortedlinkedlist.cpp

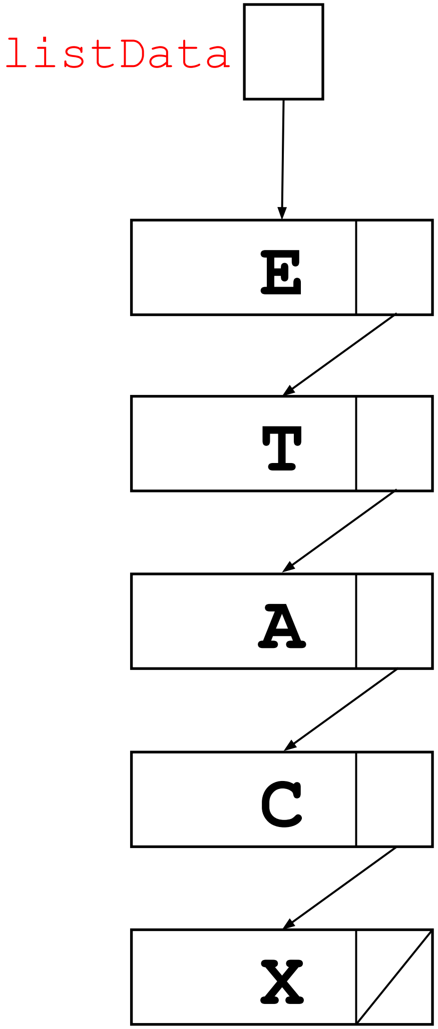
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem(‘L’)

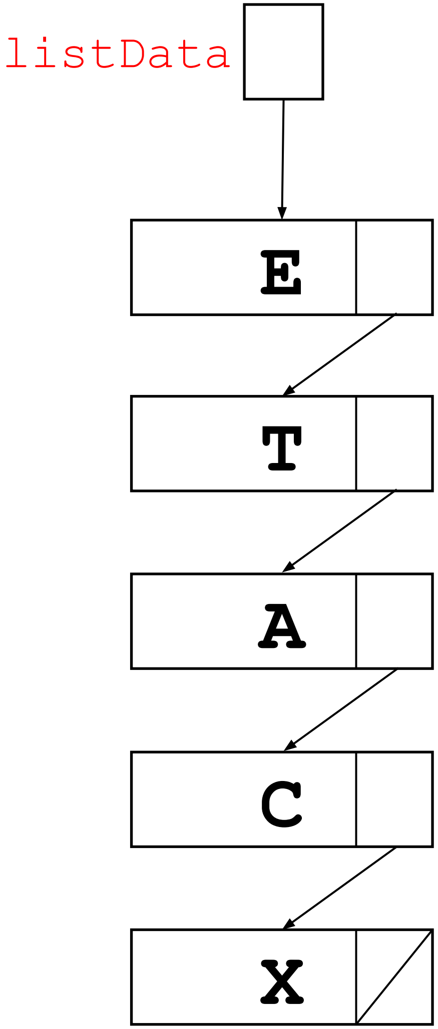
unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



unsortedlinkedlist.cpp

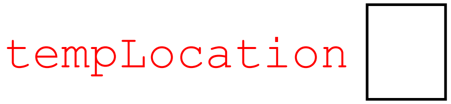
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



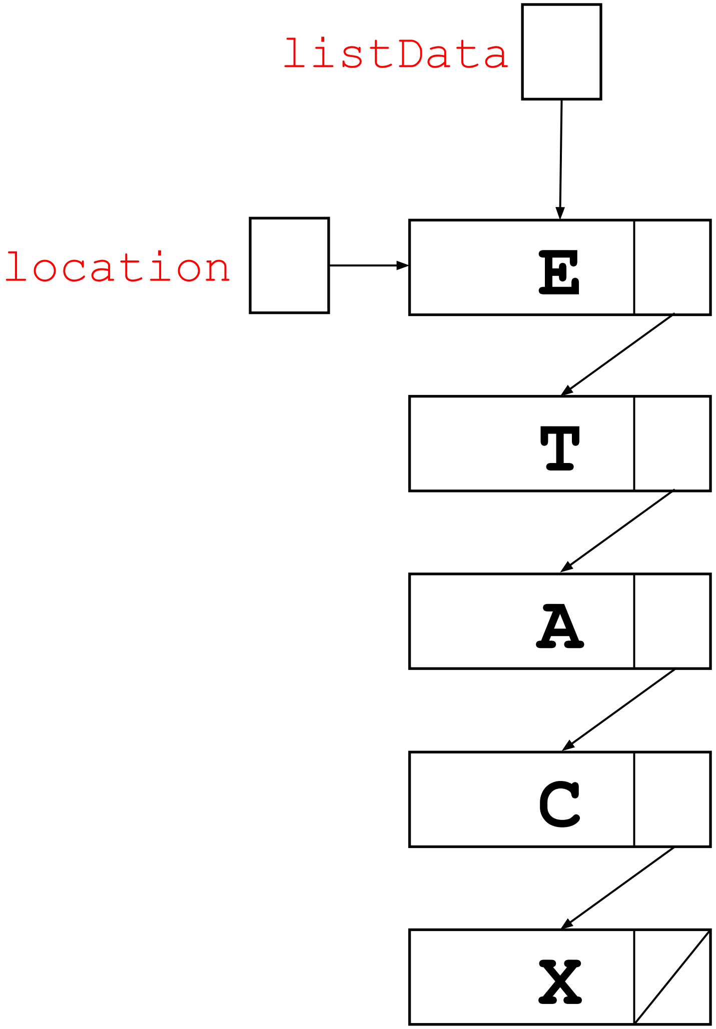
length **5**

DeleteItem('E')

unsortedlinkedlist.cpp

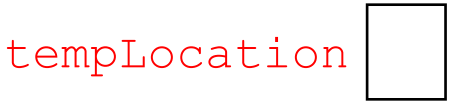


```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```

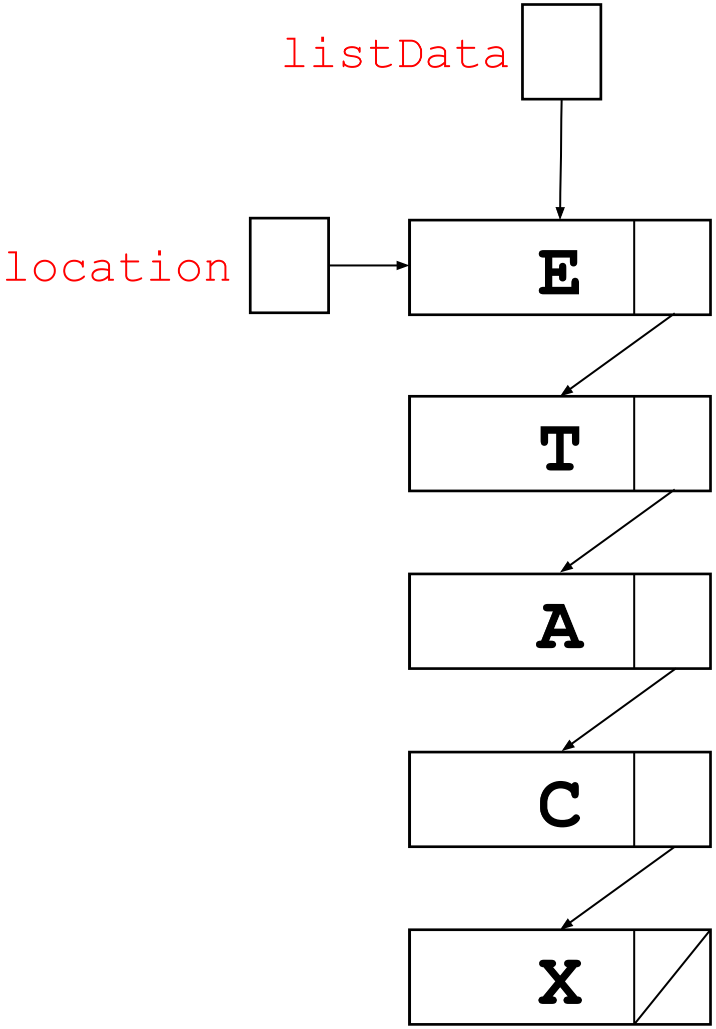


DeleteItem('E')

unsortedlinkedlist.cpp



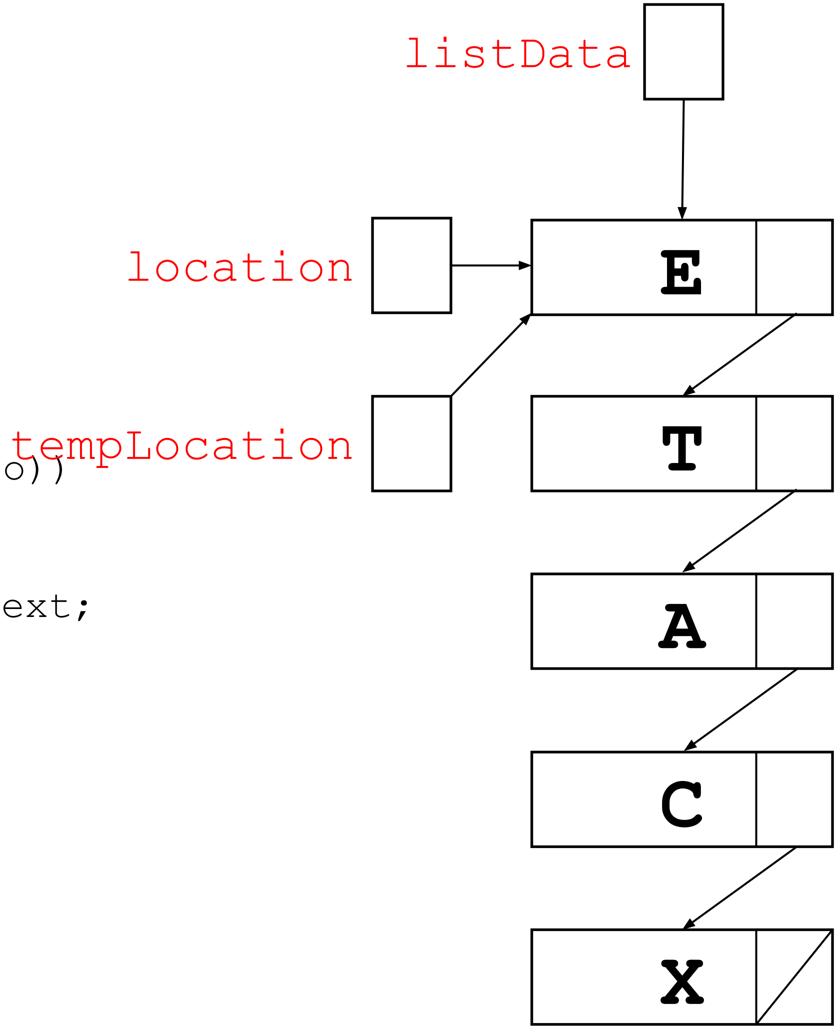
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem('E')

unsortedlinkedlist.cpp

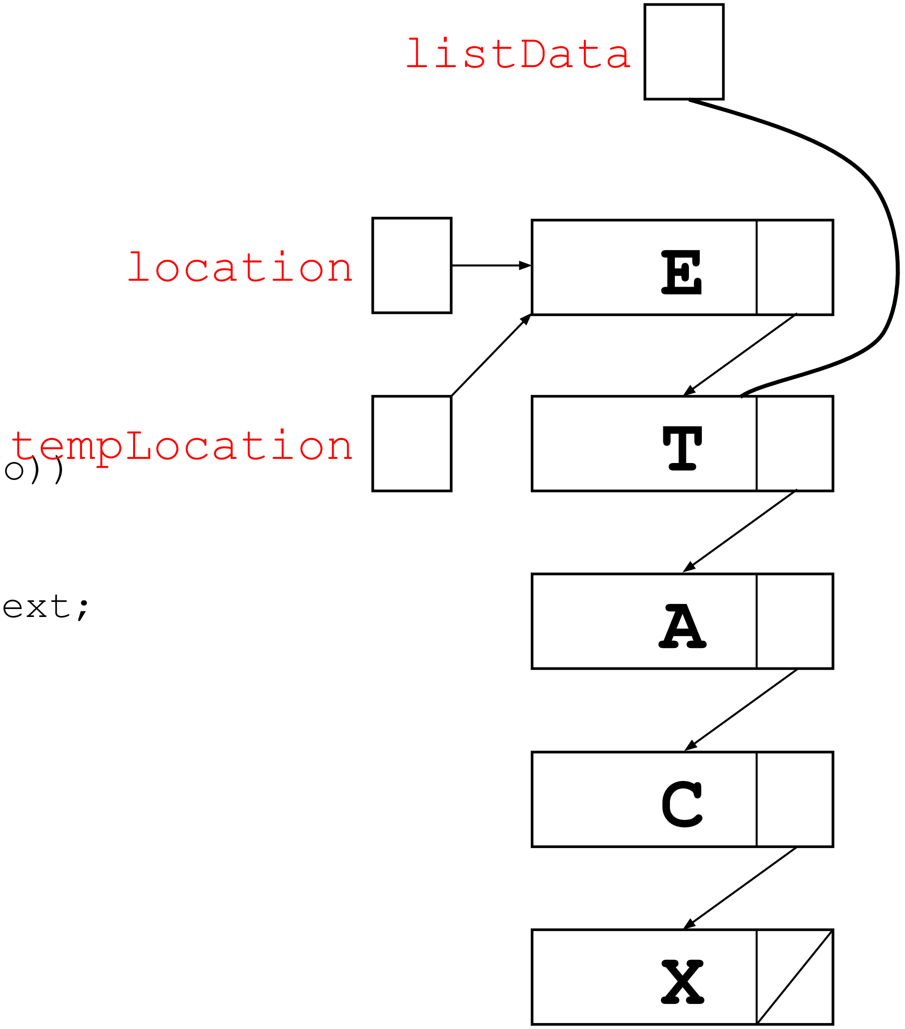
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem('E')

unsortedlinkedlist.cpp

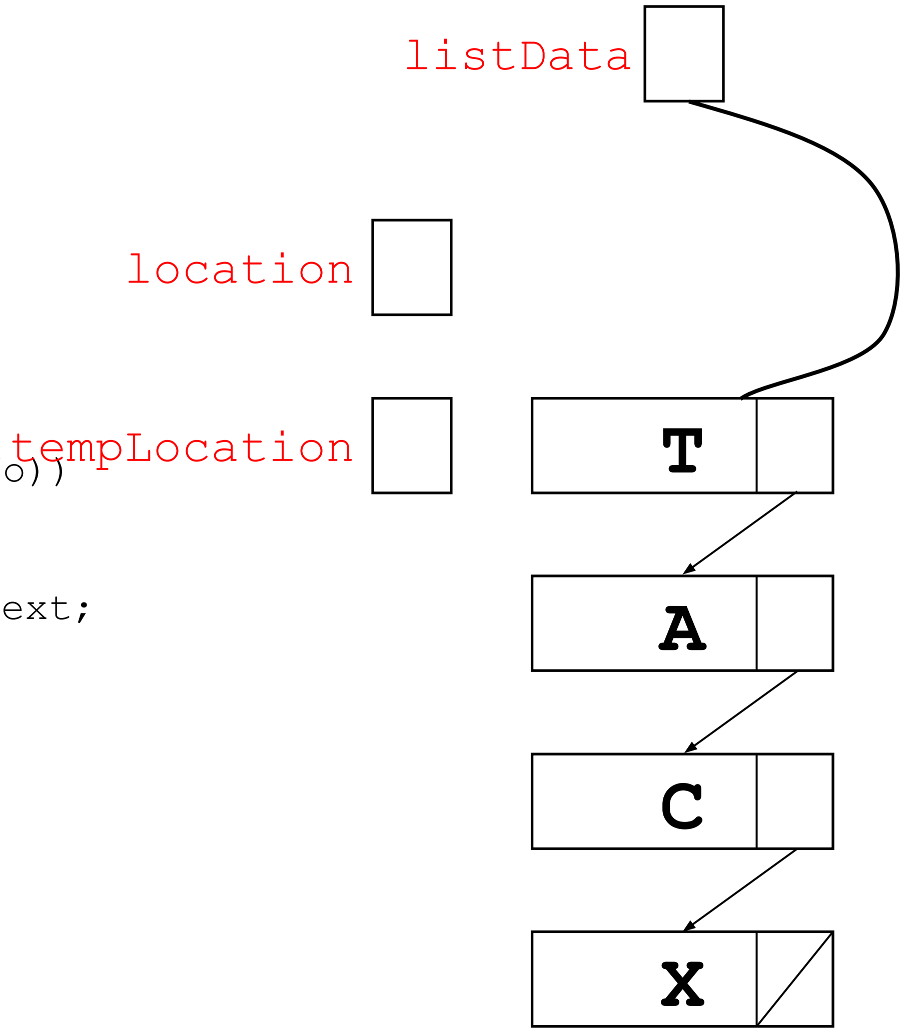
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem(‘E’)

unsortedlinkedlist.cpp

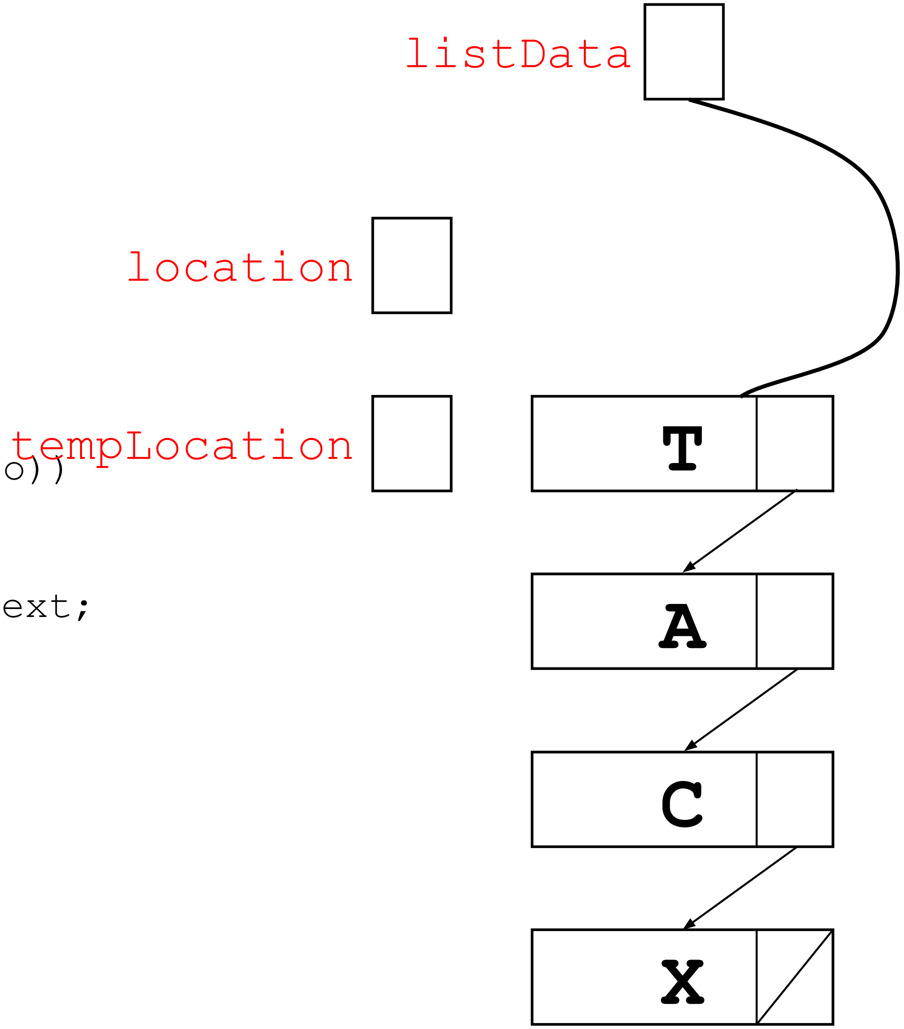
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem('E')

unsortedlinkedlist.cpp

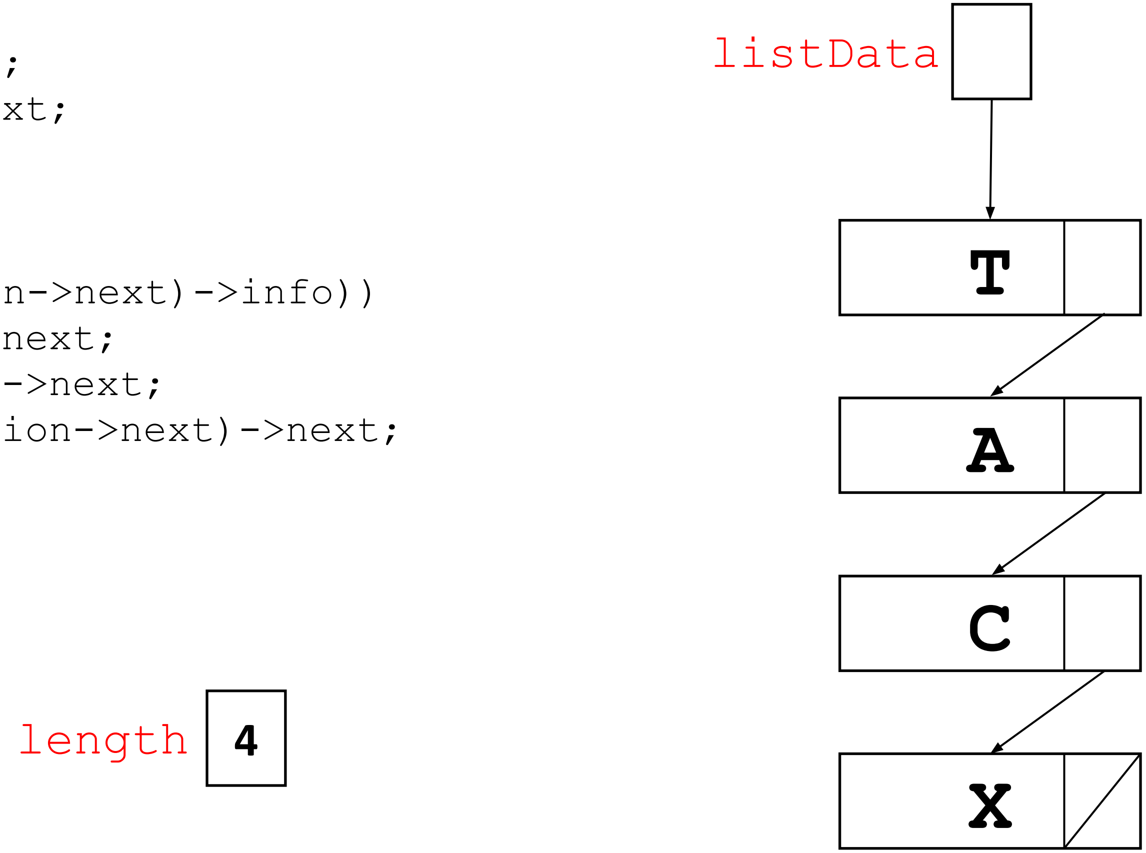
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



DeleteItem(‘E’)

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```



unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* location = listData;
  NodeType* tempLocation;
  if (item == listData->info)
  {
    tempLocation = location;
    listData = listData->next;
  }
  else
  {
    while (!(item==(location->next)->info))
      location = location->next;
    tempLocation = location->next;
    location->next = (location->next)->next;
  }
  delete tempLocation;
  length--;
}
```

O(N)

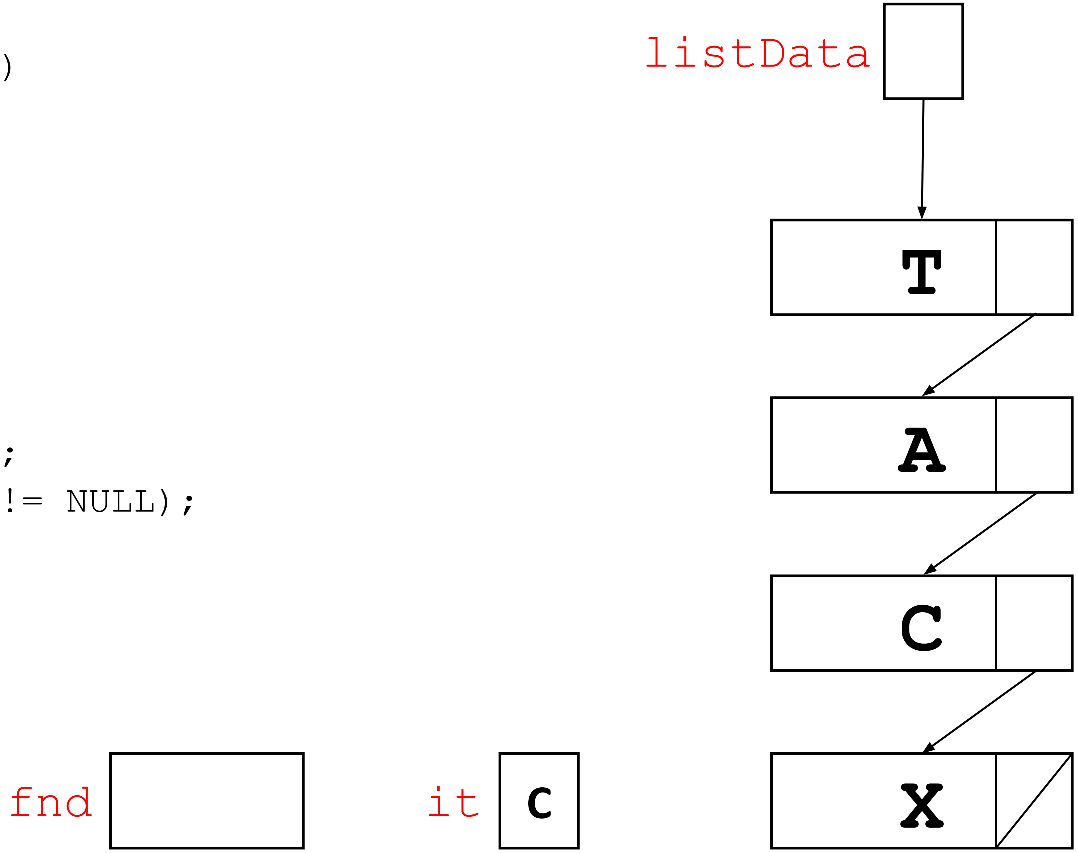
unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```

unsortedlinkedlist.cpp

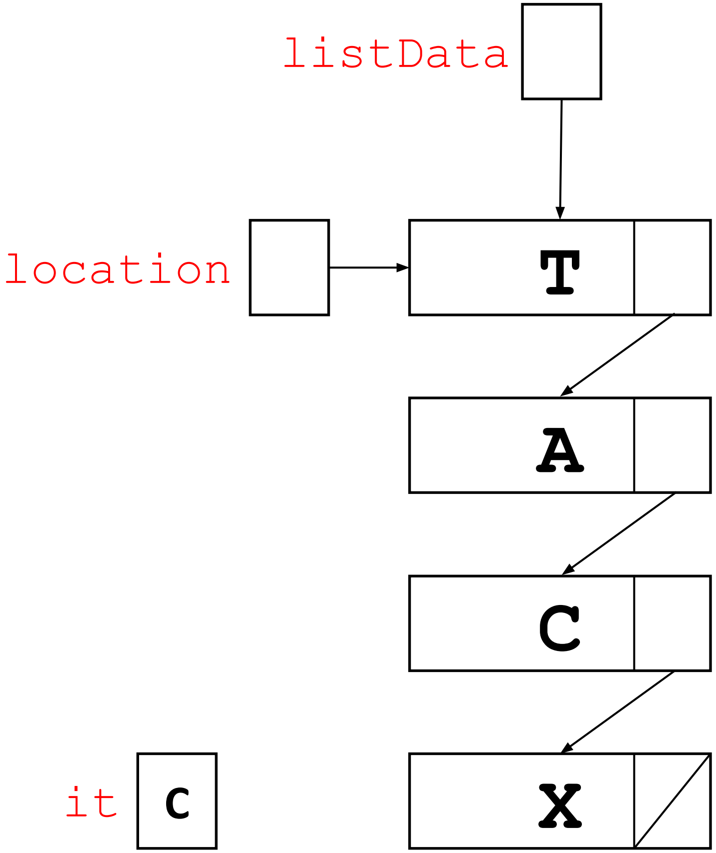
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```

RetrieveItem(it,fnd)



unsortedlinkedlist.cpp

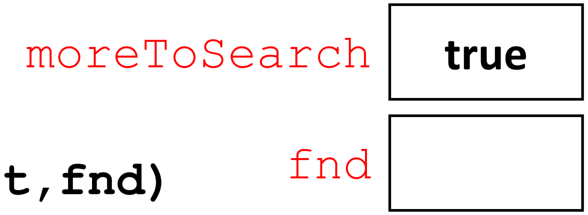
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```



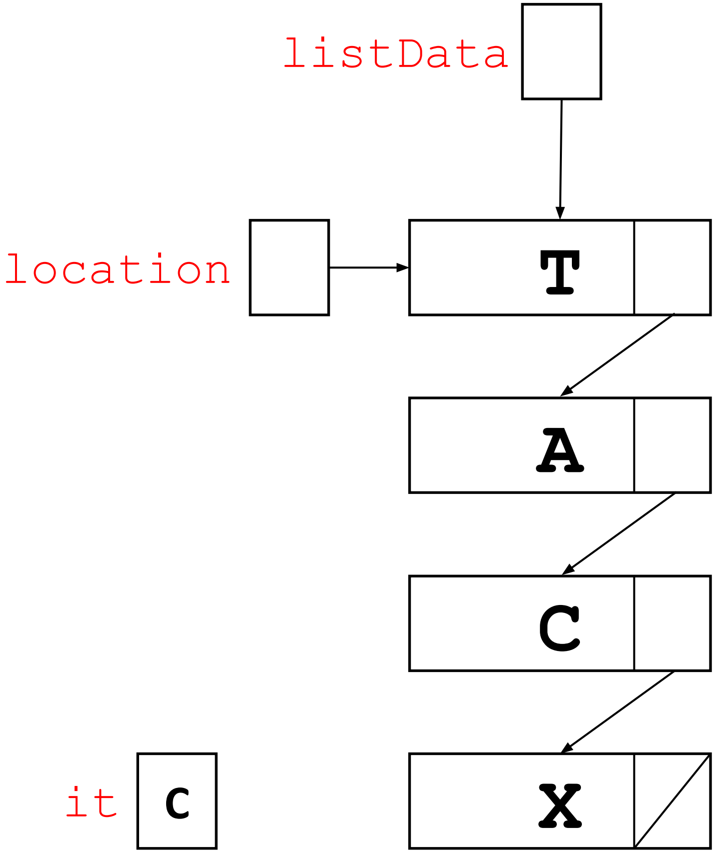
RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```

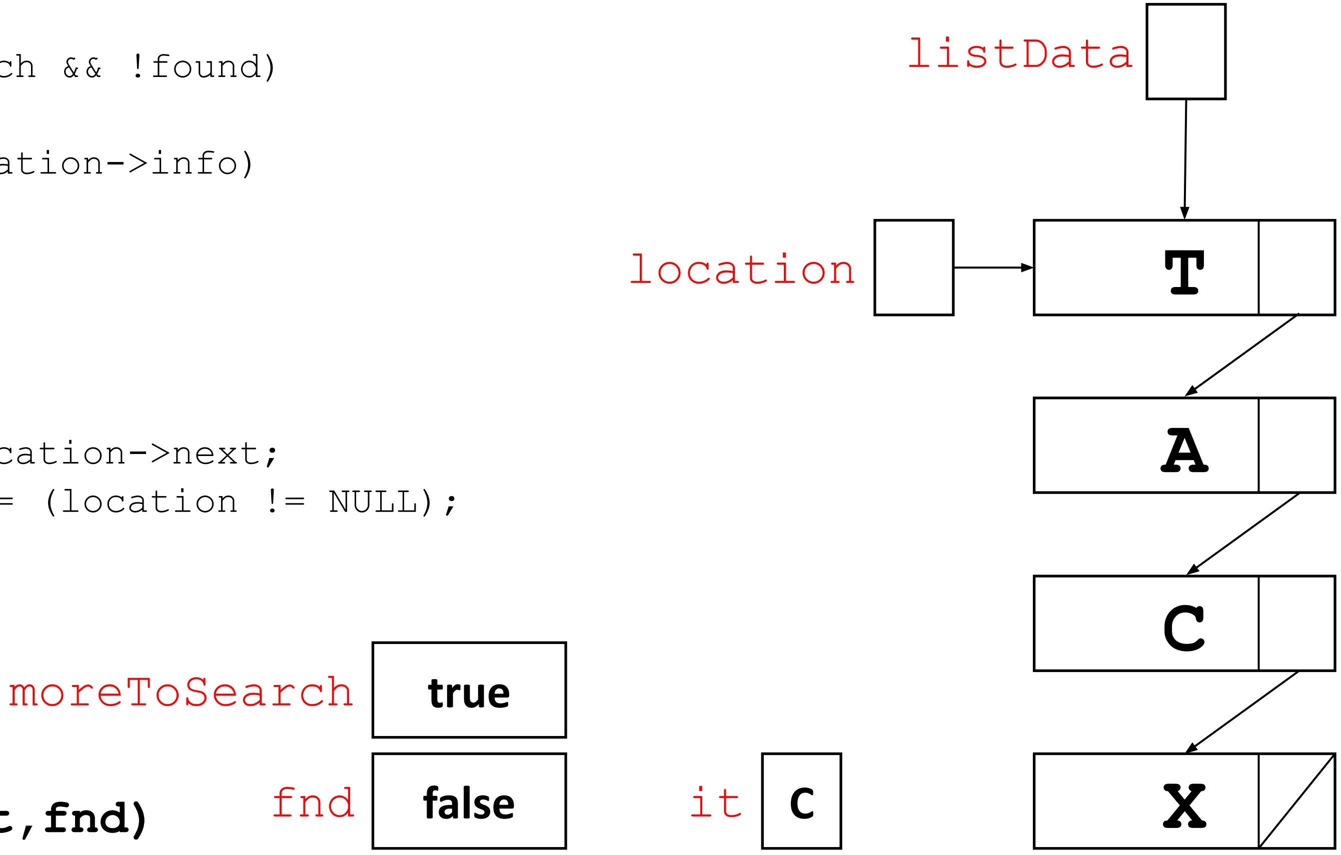


RetrieveItem(it,fnd)



unsortedlinkedlist.cpp

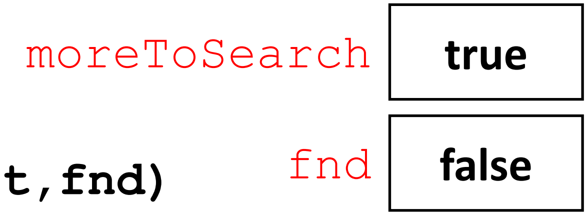
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```

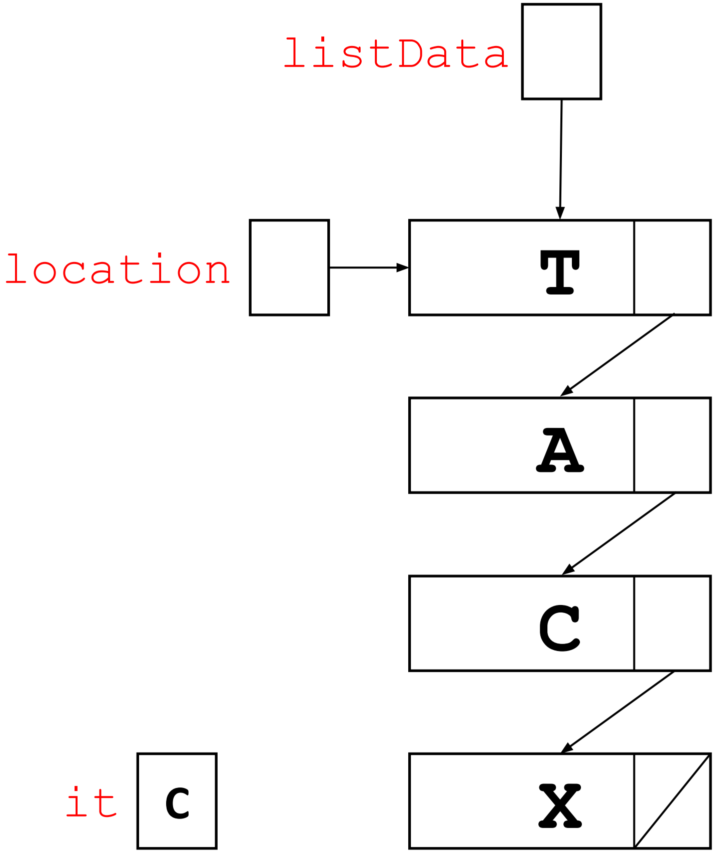


RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```



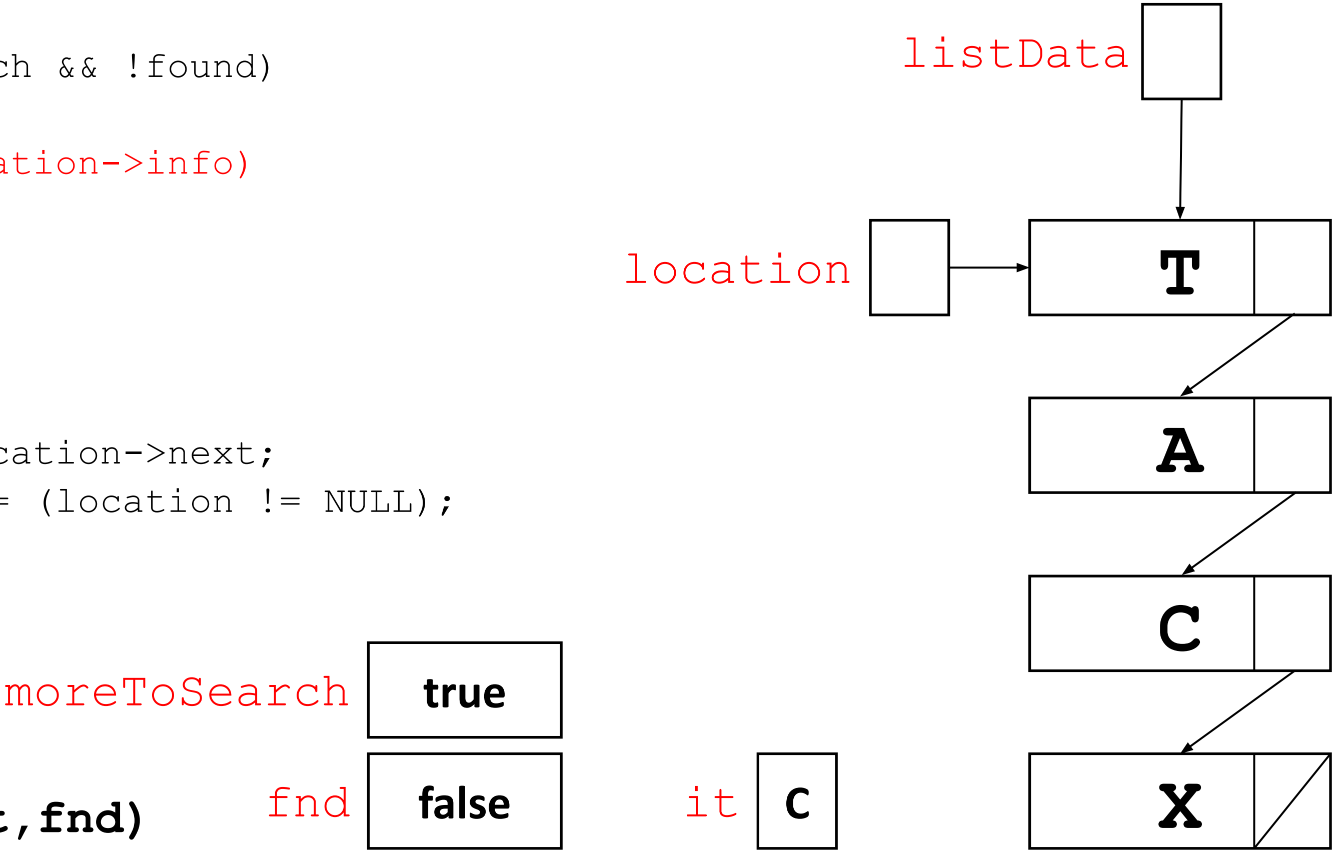


RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

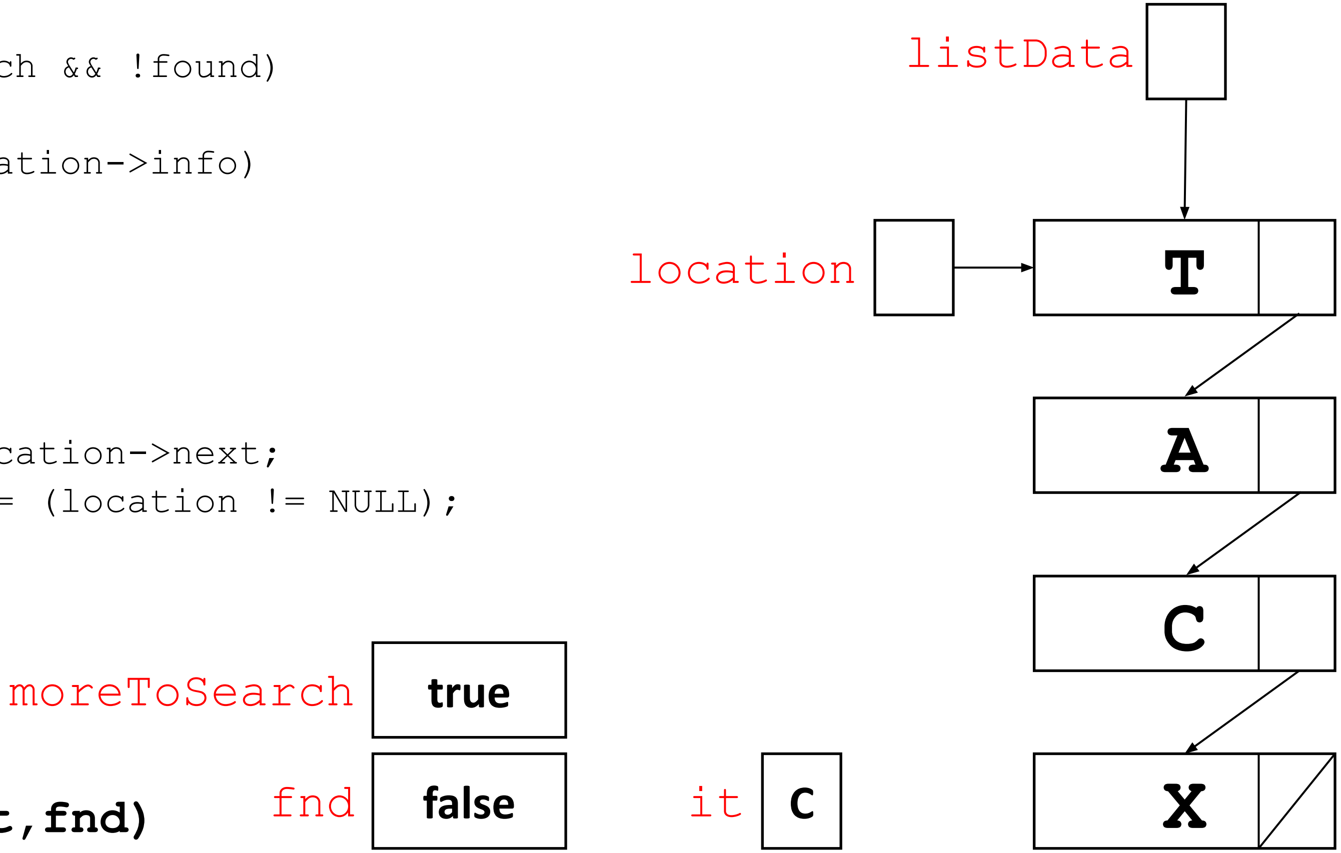
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```

RetrieveItem(it,fnd)



unsortedlinkedlist.cpp

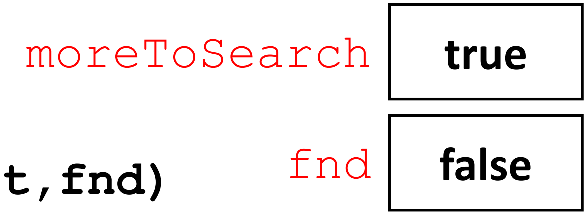
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```

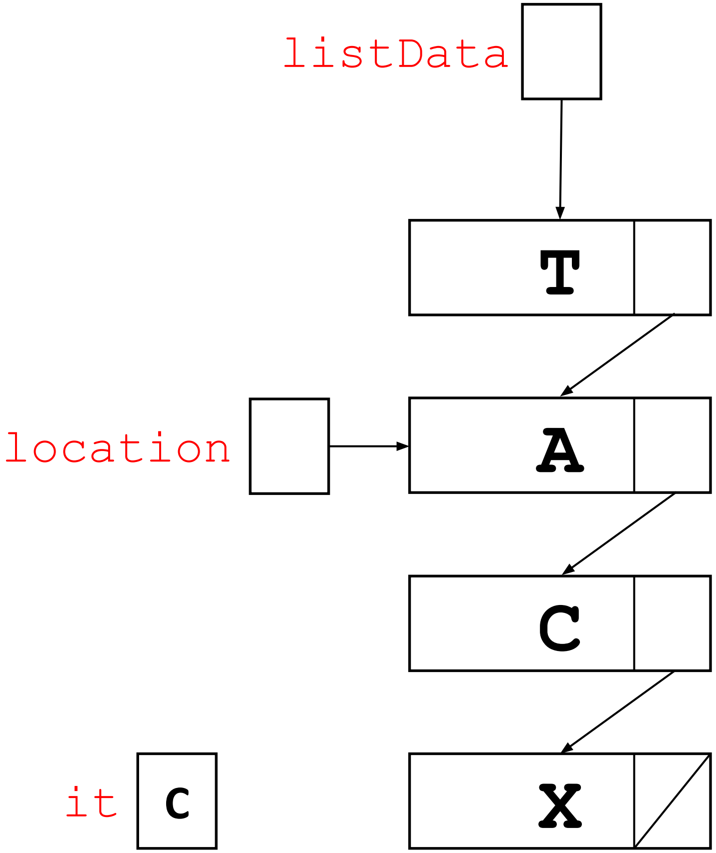


RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```

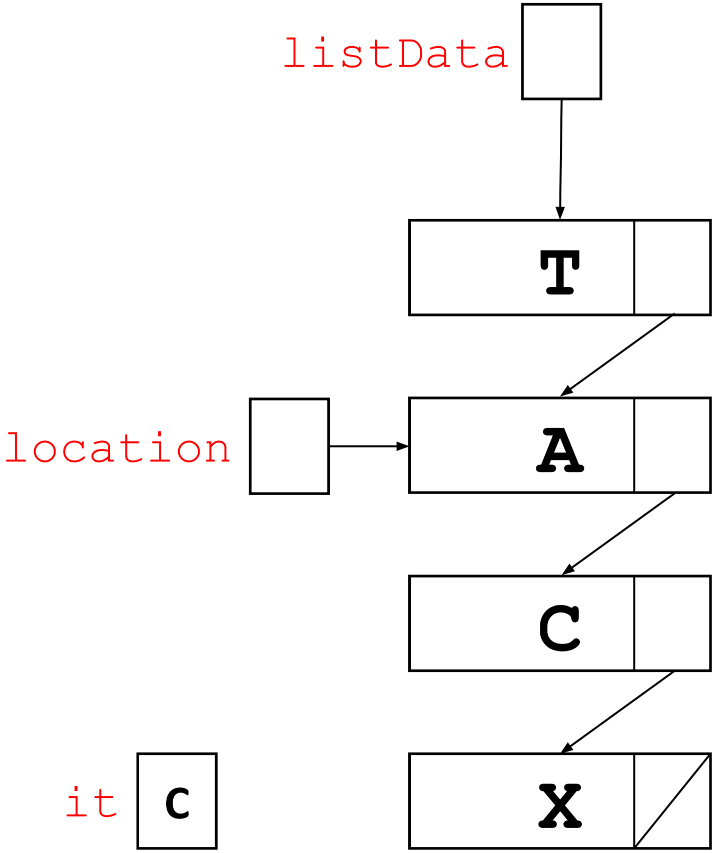




RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```



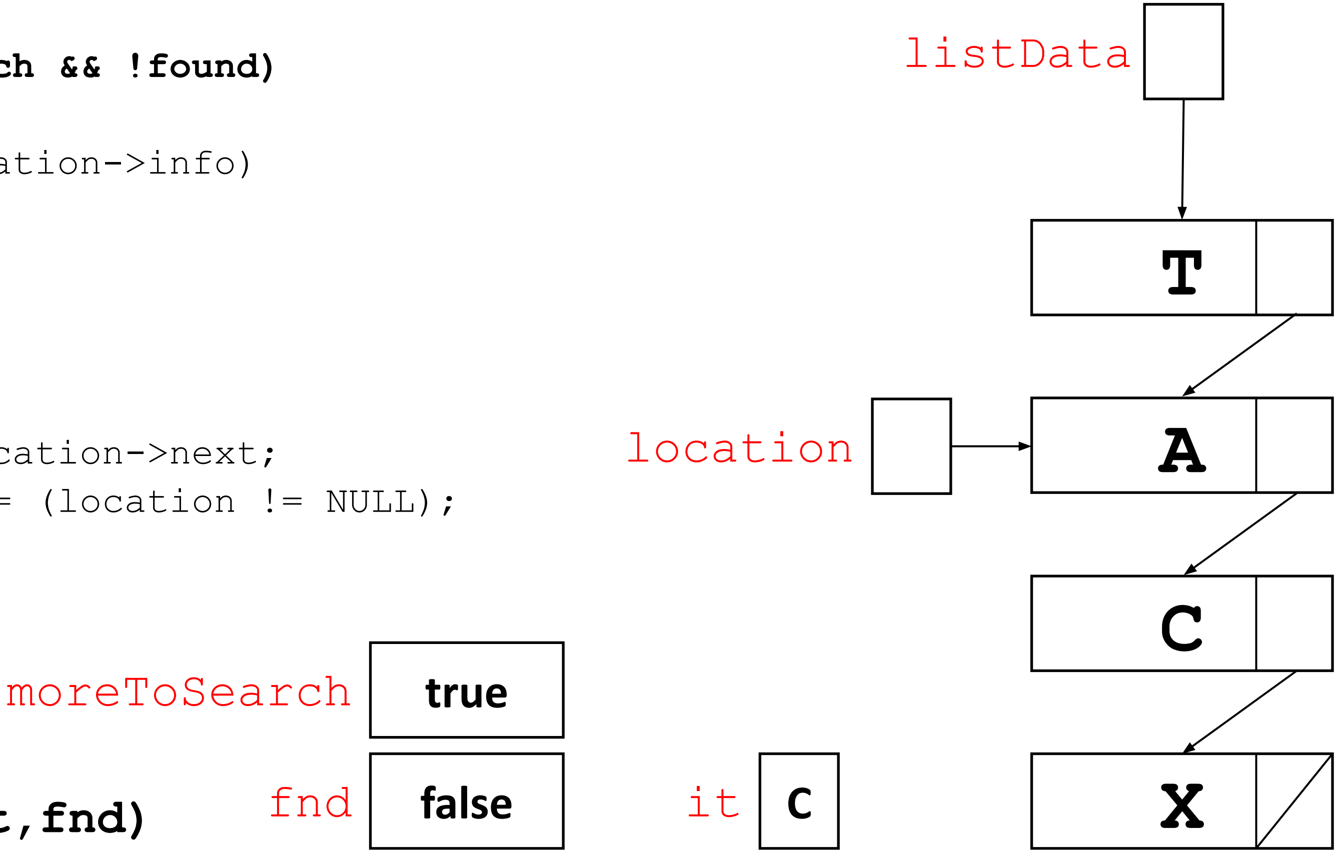
moreToSearch **true**

fnd **false**

RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```



RetrieveItem(it,fnd)

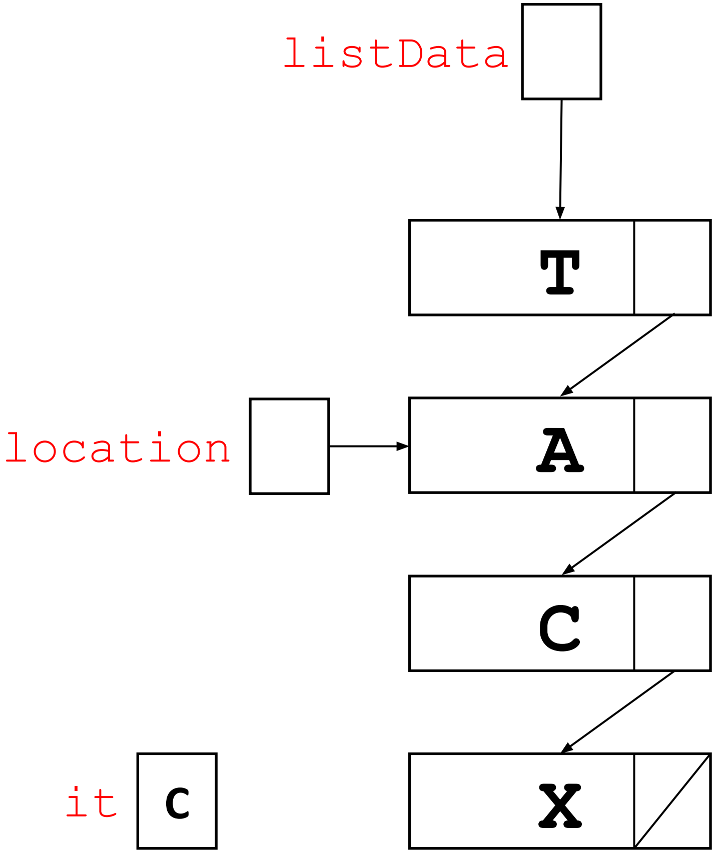
unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```

moreToSearch **true**

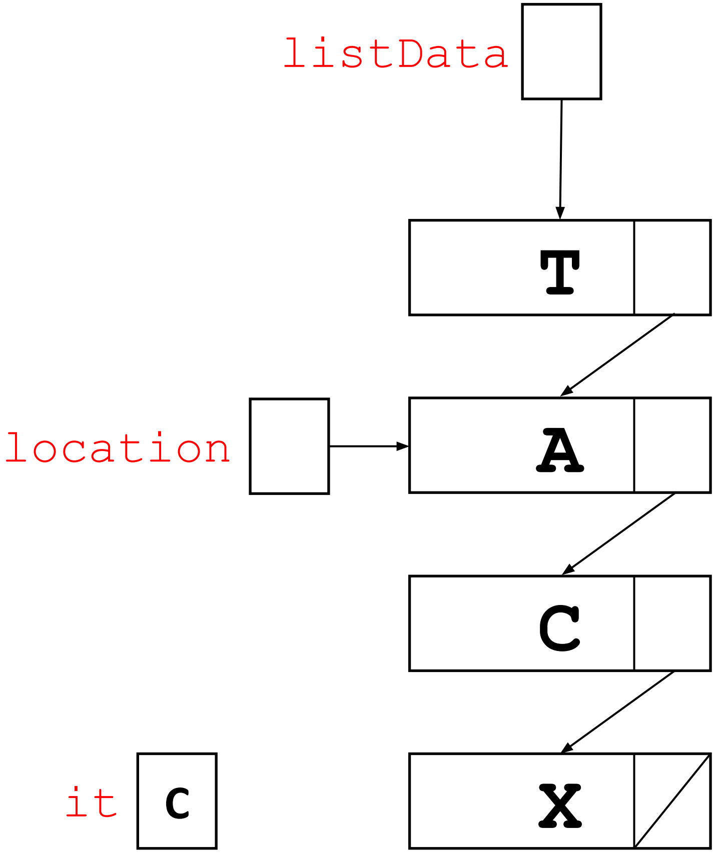
fnd **false**

RetrieveItem(it,fnd)



unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```



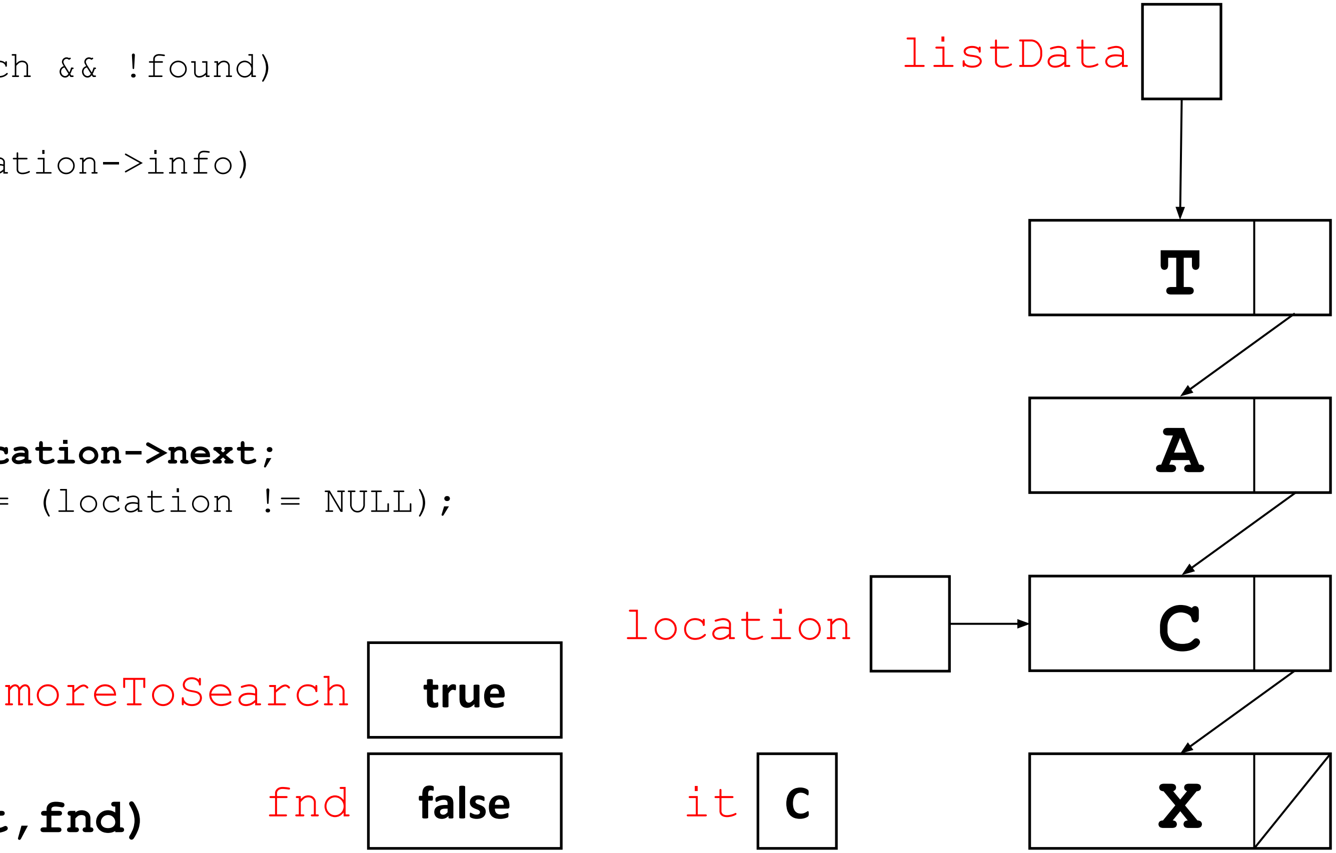
moreToSearch **true**

fnd **false**

RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

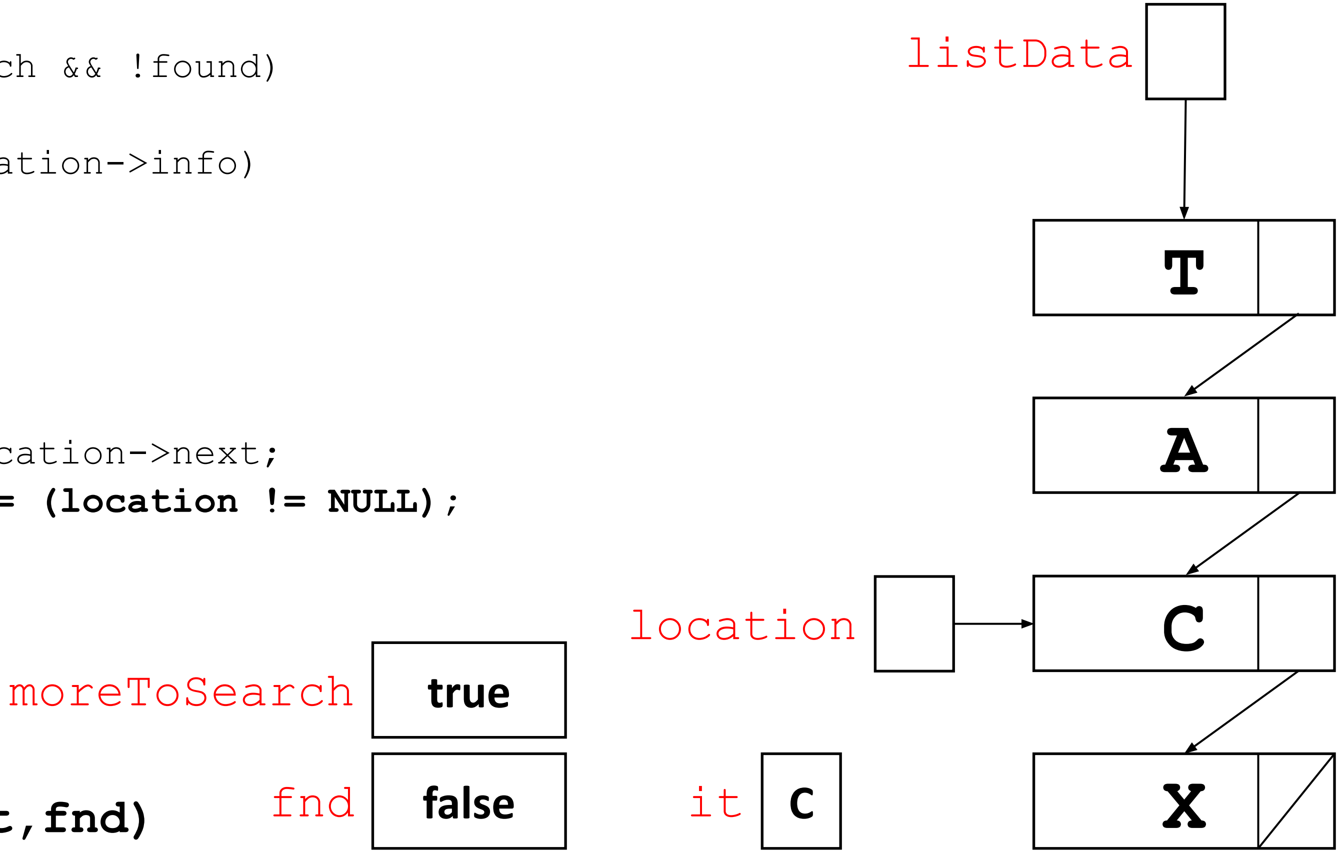
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```



RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

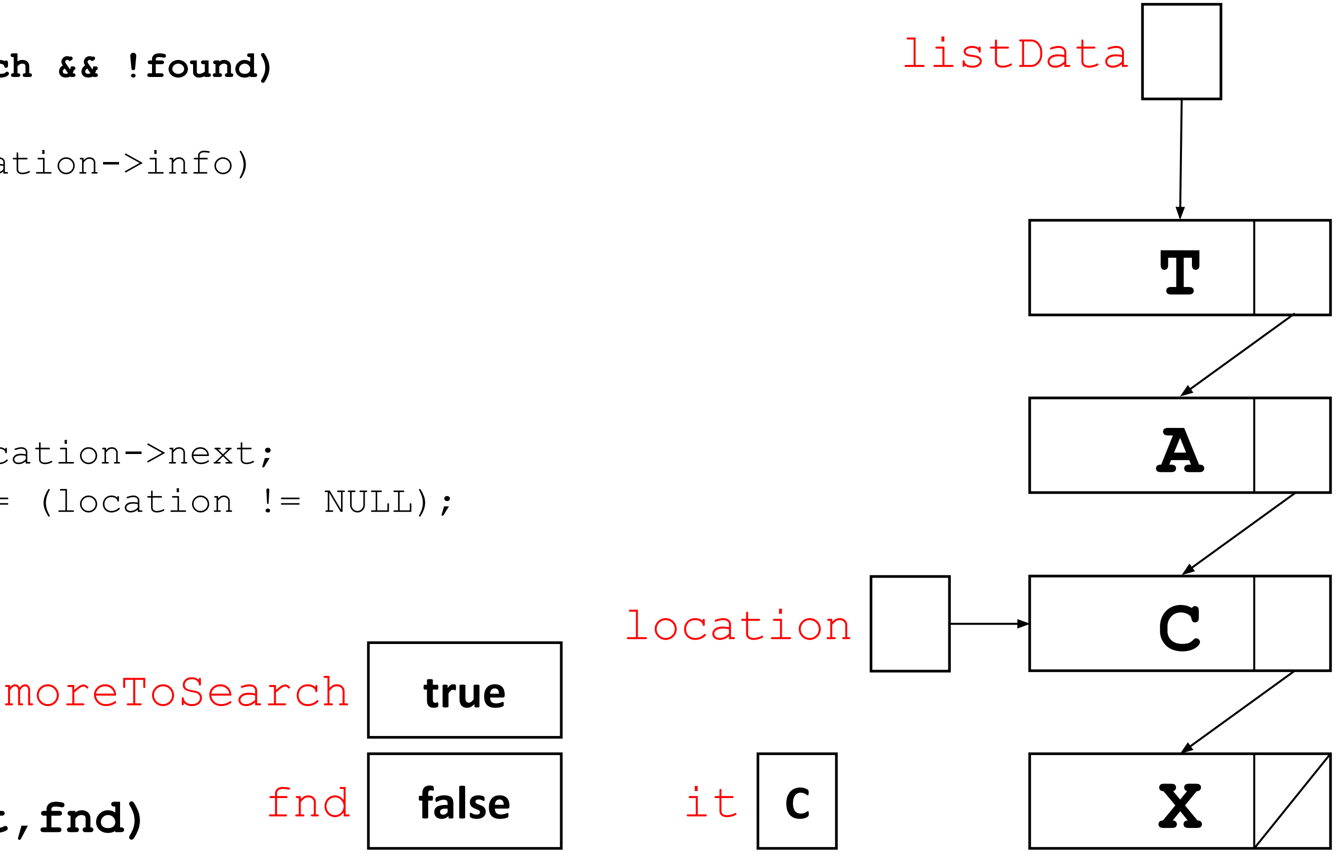
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```



RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

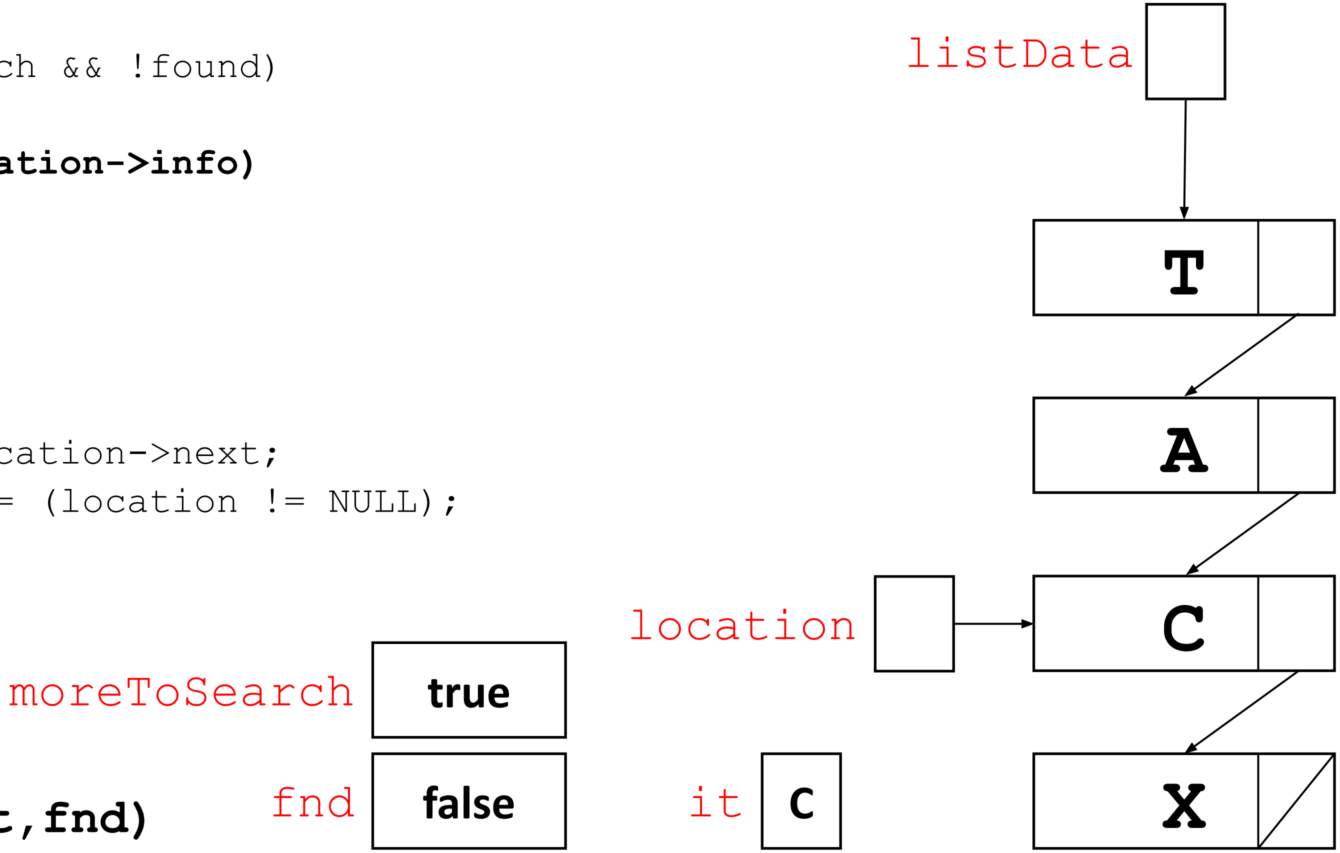
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```



RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

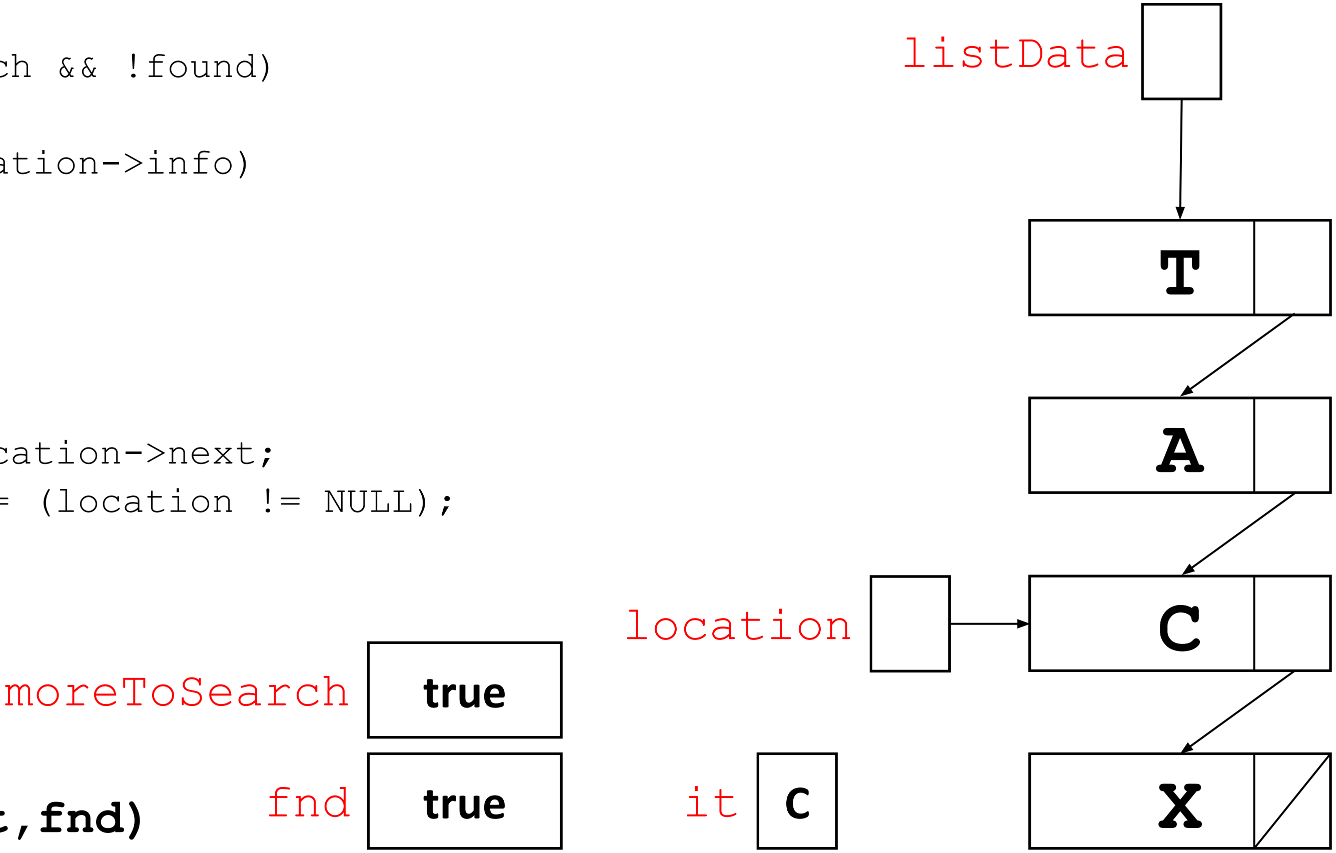
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```



RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

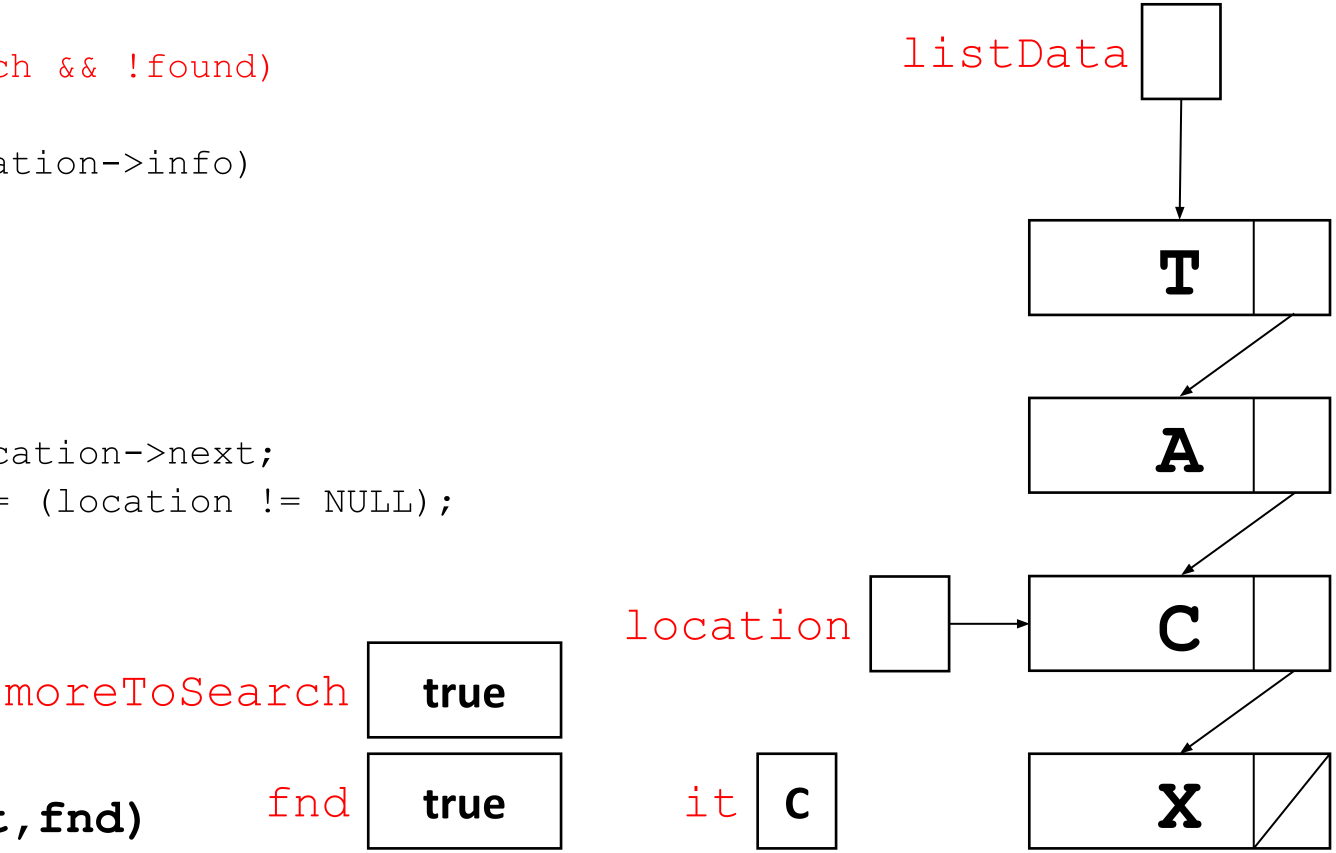
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```



RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

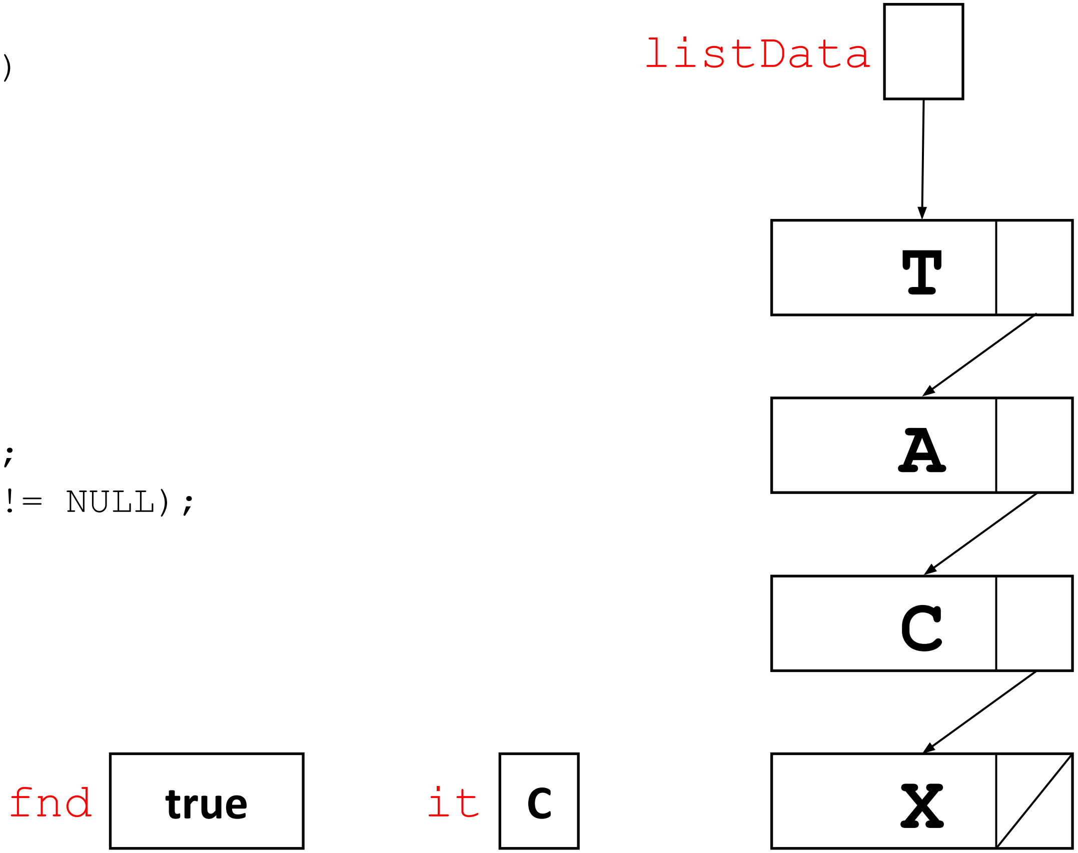
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```



RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```



unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* location = listData;
  bool moreToSearch = (location != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == location->info)
    {
      found = true;
    }
    else
    {
      location = location->next;
      moreToSearch = (location != NULL);
    }
  }
}
```

O(N)

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::MakeEmpty()
{
  NodeType* tempPtr;

  while (listData != NULL)
  {
    tempPtr = listData;
    listData = listData->next;
    delete tempPtr;
  }
  length = 0;
}

template <class ItemType>
UnsortedType<ItemType>::~UnsortedType()
{
  MakeEmpty();
}
```

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::MakeEmpty()
{
  NodeType* tempPtr;

  while (listData != NULL)
  {
    tempPtr = listData;
    listData = listData->next;
    delete tempPtr;
  }
  length = 0;
}
```

O(N)

```
template <class ItemType>
UnsortedType<ItemType>::~UnsortedType()
{
  MakeEmpty();
}
```

O(N)

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::ResetList()
{
  currentPos = NULL;
}

template <class ItemType>
void UnsortedType<ItemType>::GetNextItem(ItemType& item)
{
  if (currentPos == NULL)
    currentPos = listData;
  else
    currentPos = currentPos->next;
  item = currentPos->info;
}
```

unsortedlinkedlist.cpp

```
template <class ItemType>
void UnsortedType<ItemType>::ResetList()
{
  currentPos = NULL;
}
```

O(1)

```
template <class ItemType>
void UnsortedType<ItemType>::GetNextItem(ItemType& item)
{
  if (currentPos == NULL)
    currentPos = listData;
  else
    currentPos = currentPos->next;
  item = currentPos->info;
}
```

O(1)