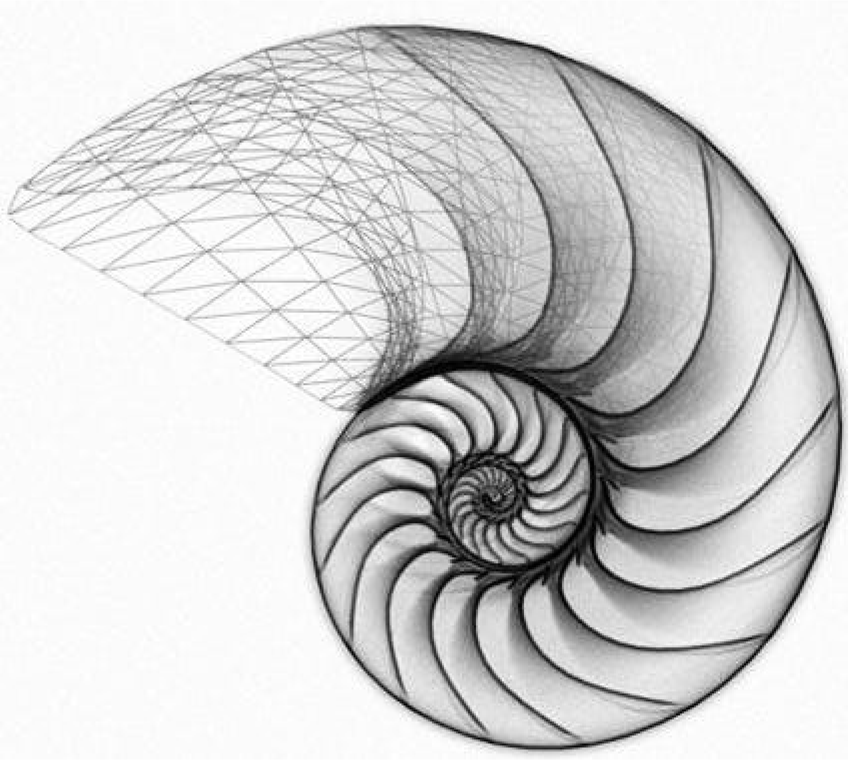
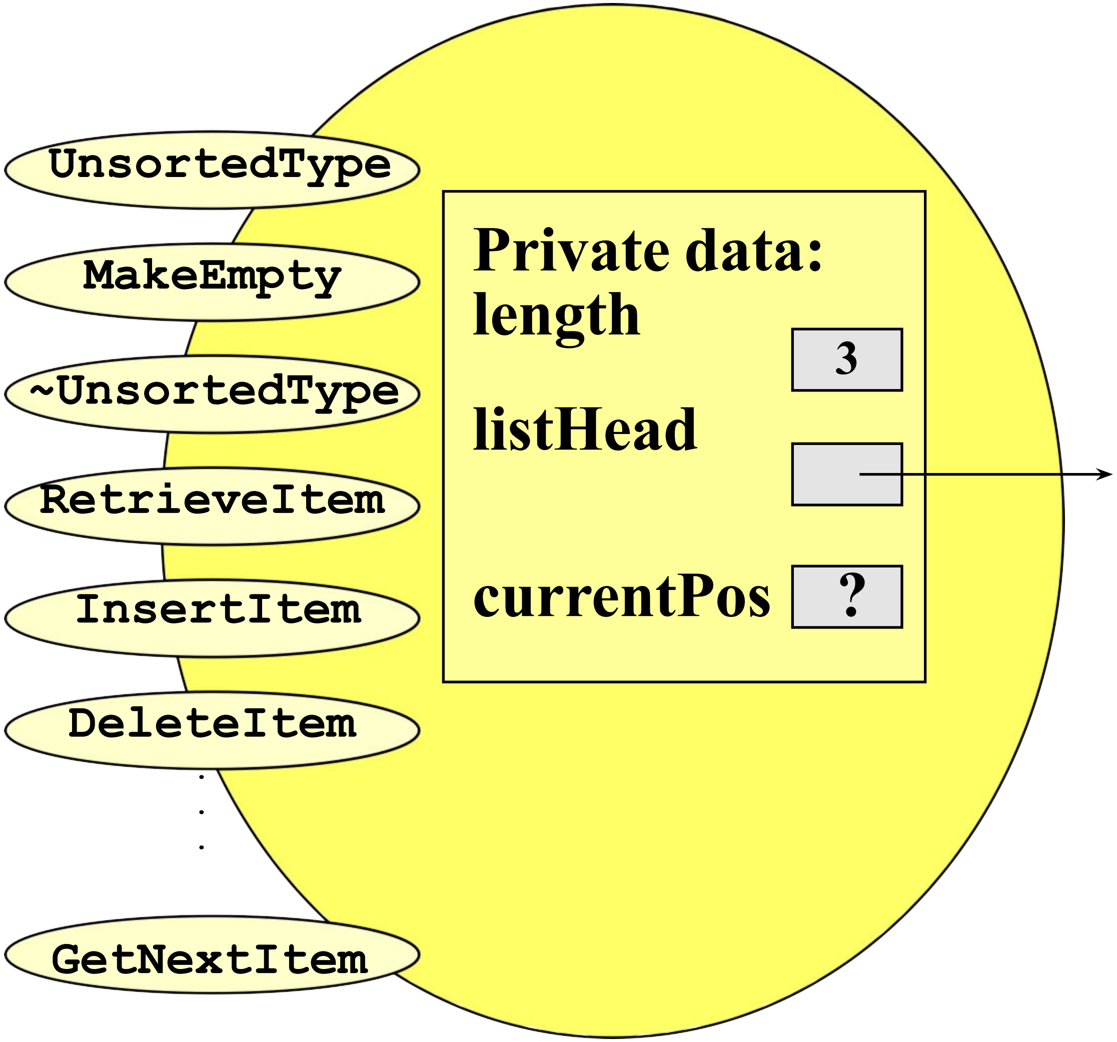


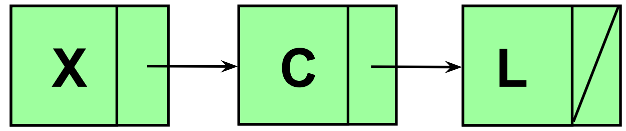
Lecture 10

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

CSE225: Data Structures and Algorithms

class UnsortedType<char>





Addr.	Info.	Next Addr.
1	Alu	4
2	Potol	3
3	Morich	Null
4	Piyaj	3
5	Holud	1

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* listHead;
  int length;
  NodeType* currentPos;
};

#endif // UNSORTEDLINKEDLIST_H_INCLUDED
```

unsortedlinkedlist.cpp

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

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

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

```cpp
template<class ItemType>
bool UnsortedType<ItemType>::IsFull()
{
  NodeType* position;
  try
  {
    position = new NodeType;
    delete position;
    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;
  listHead = NULL;
  currentPos = NULL;
}
```

O(1)

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

O(1)

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

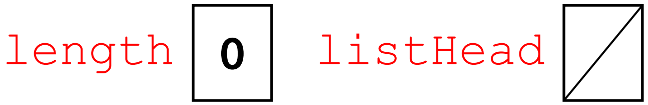
O(1)

unsortedlinkedlist.cpp

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

unsortedlinkedlist.cpp

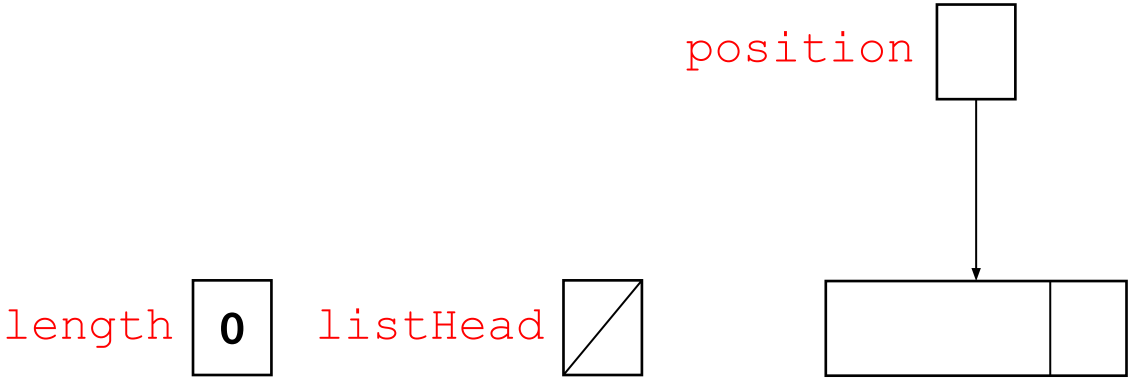
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('X')

unsortedlinkedlist.cpp

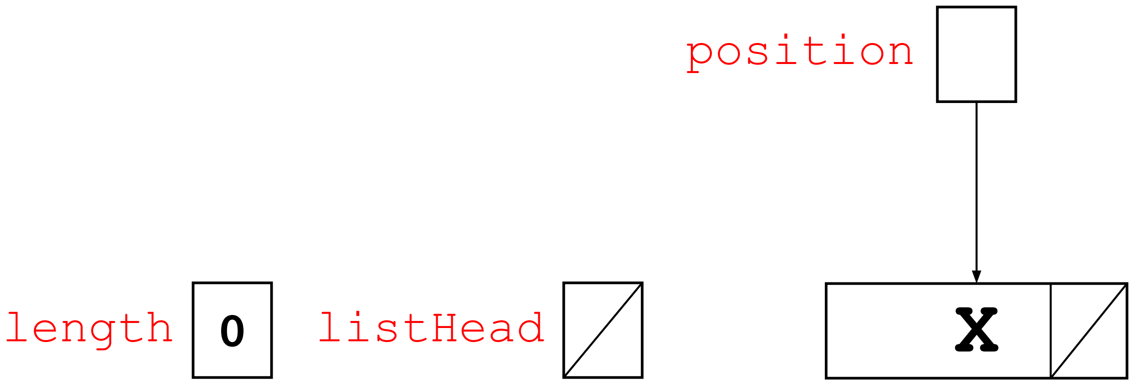
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('X')

unsortedlinkedlist.cpp

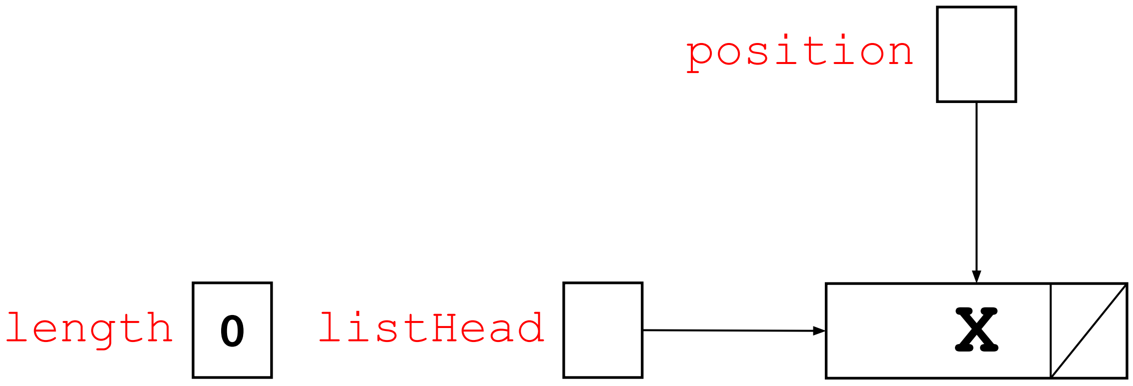
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('X')

unsortedlinkedlist.cpp

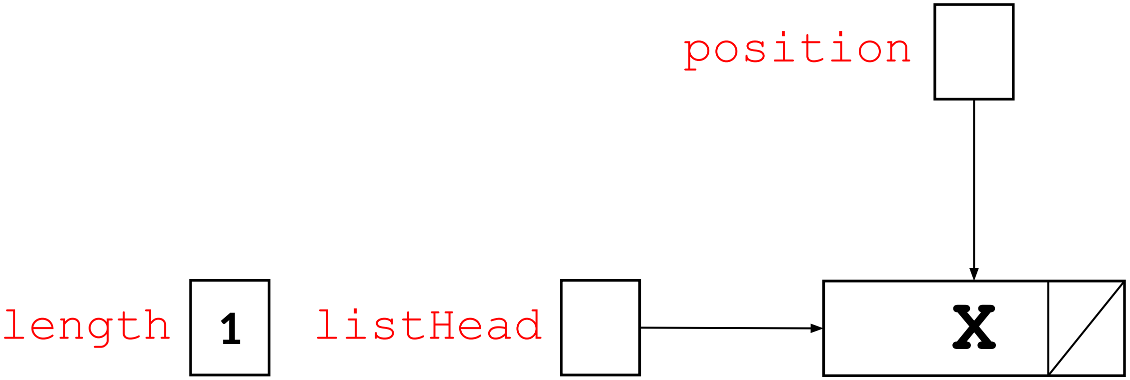
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('X')

unsortedlinkedlist.cpp

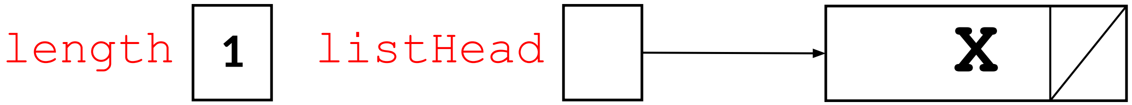
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('X')

unsortedlinkedlist.cpp

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

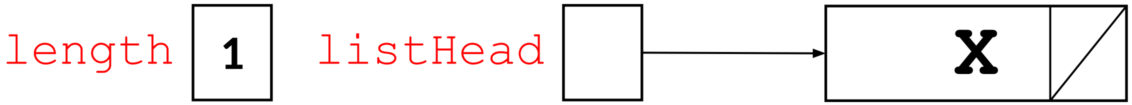


length 1 listHead

X

unsortedlinkedlist.cpp

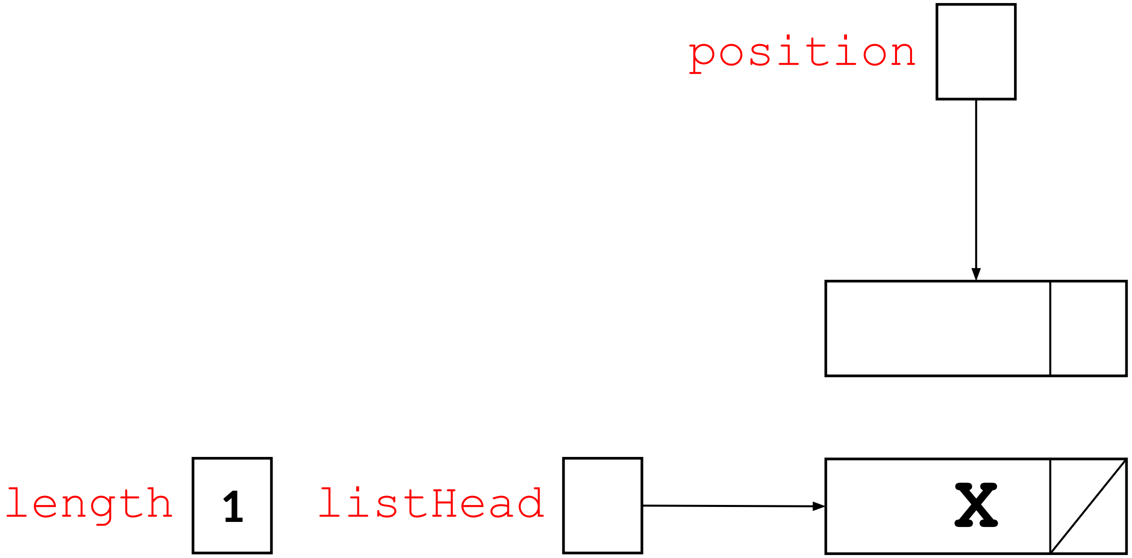
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('C')

unsortedlinkedlist.cpp

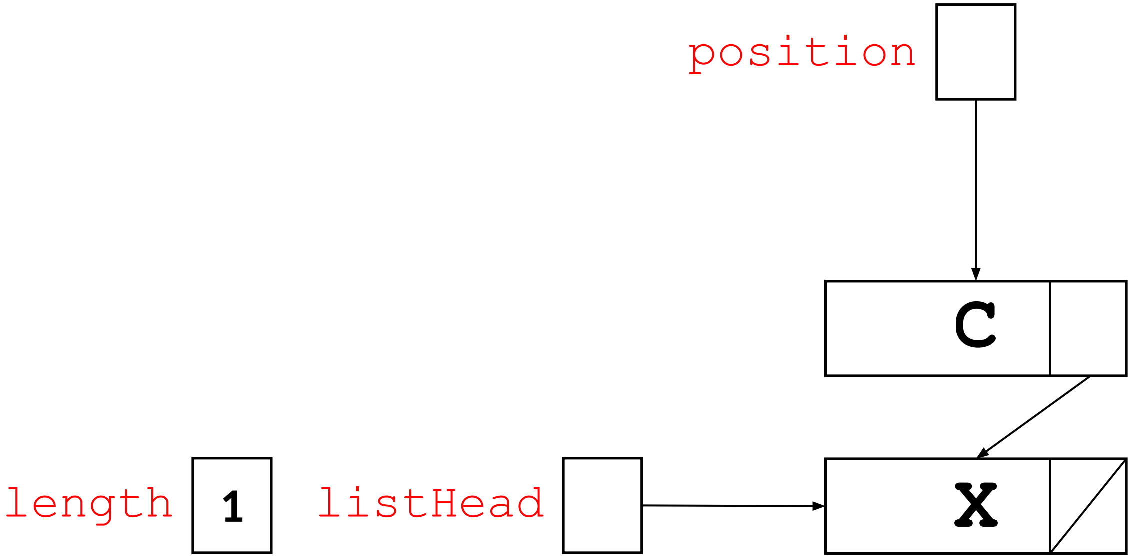
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('C')

unsortedlinkedlist.cpp

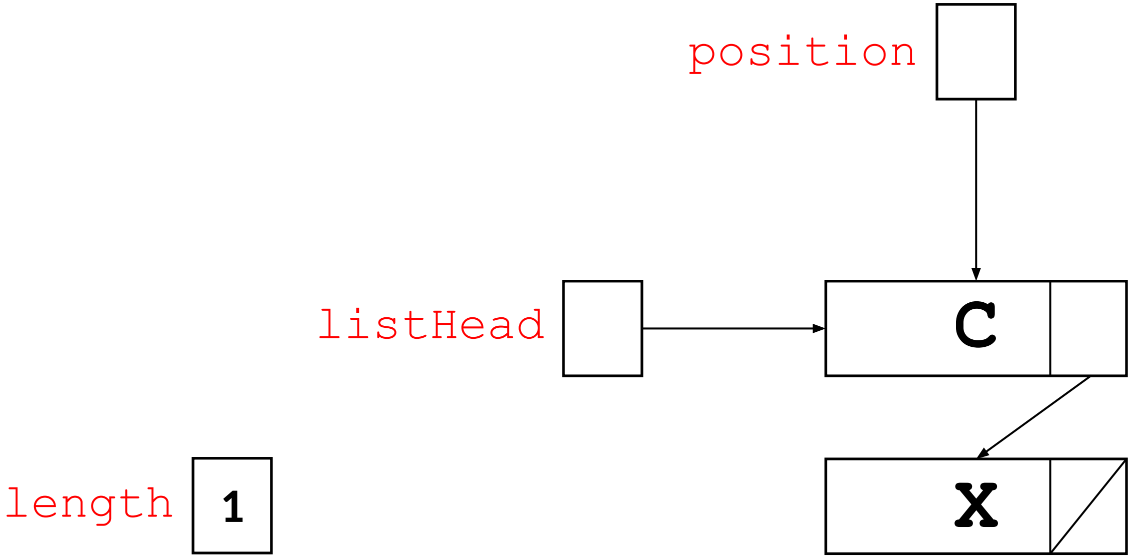
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('C')

unsortedlinkedlist.cpp

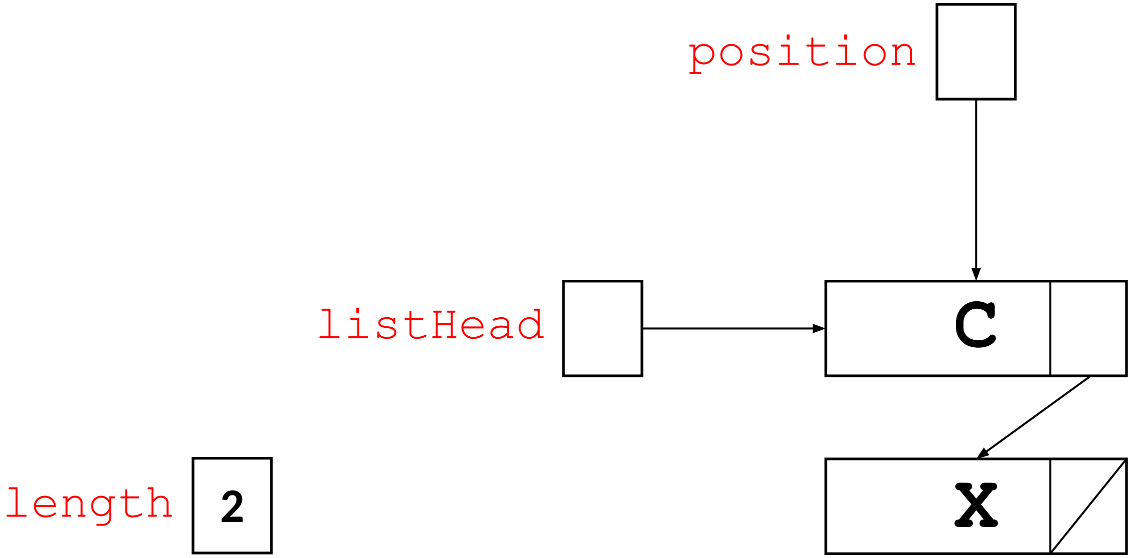
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('C')

unsortedlinkedlist.cpp

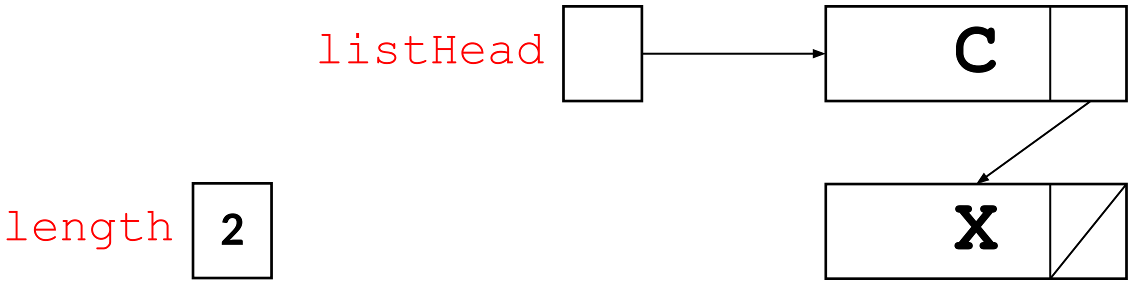
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('C')

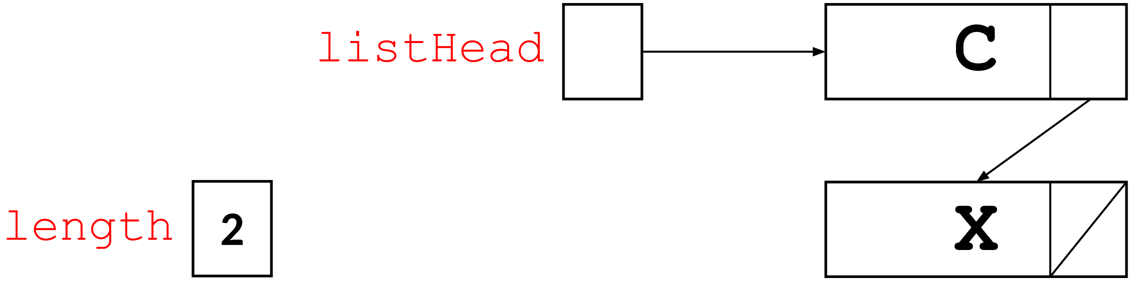
unsortedlinkedlist.cpp

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



unsortedlinkedlist.cpp

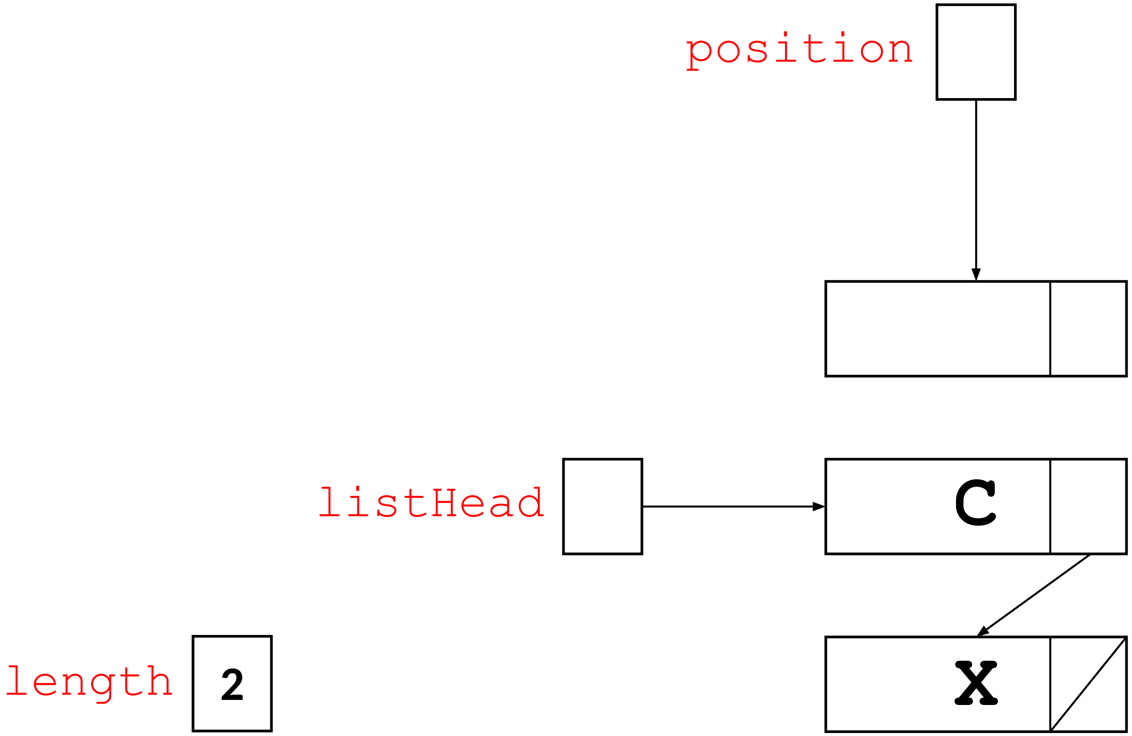
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('L')

unsortedlinkedlist.cpp

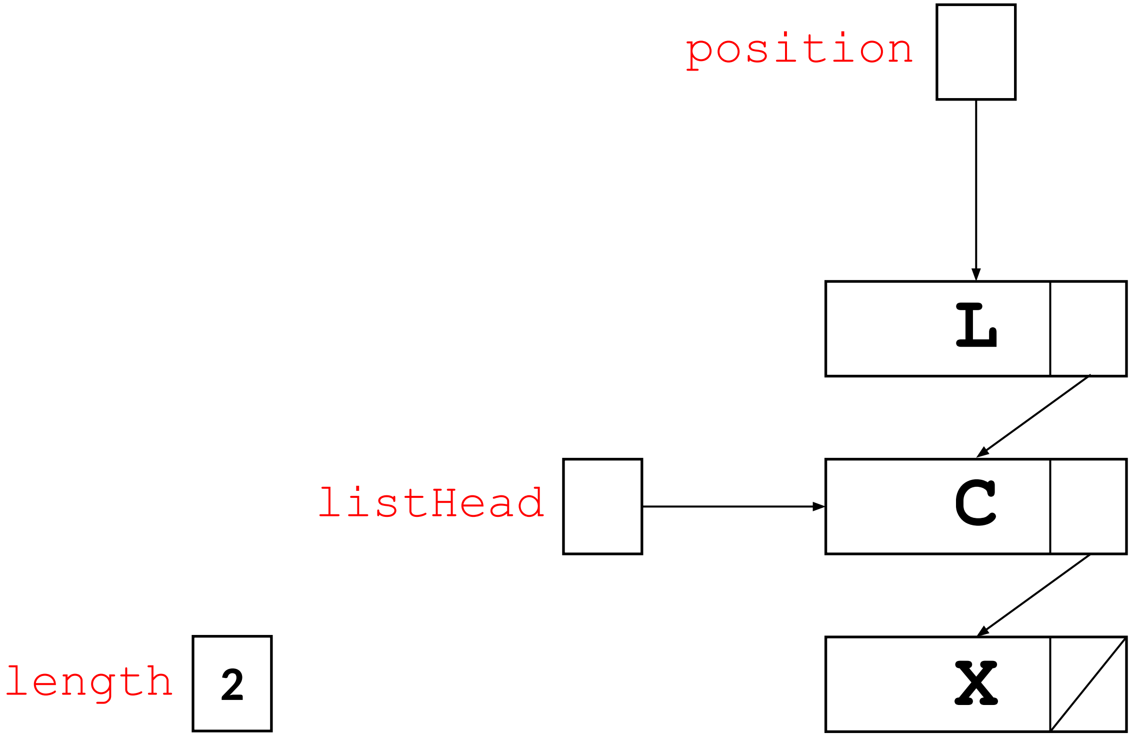
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('L')

unsortedlinkedlist.cpp

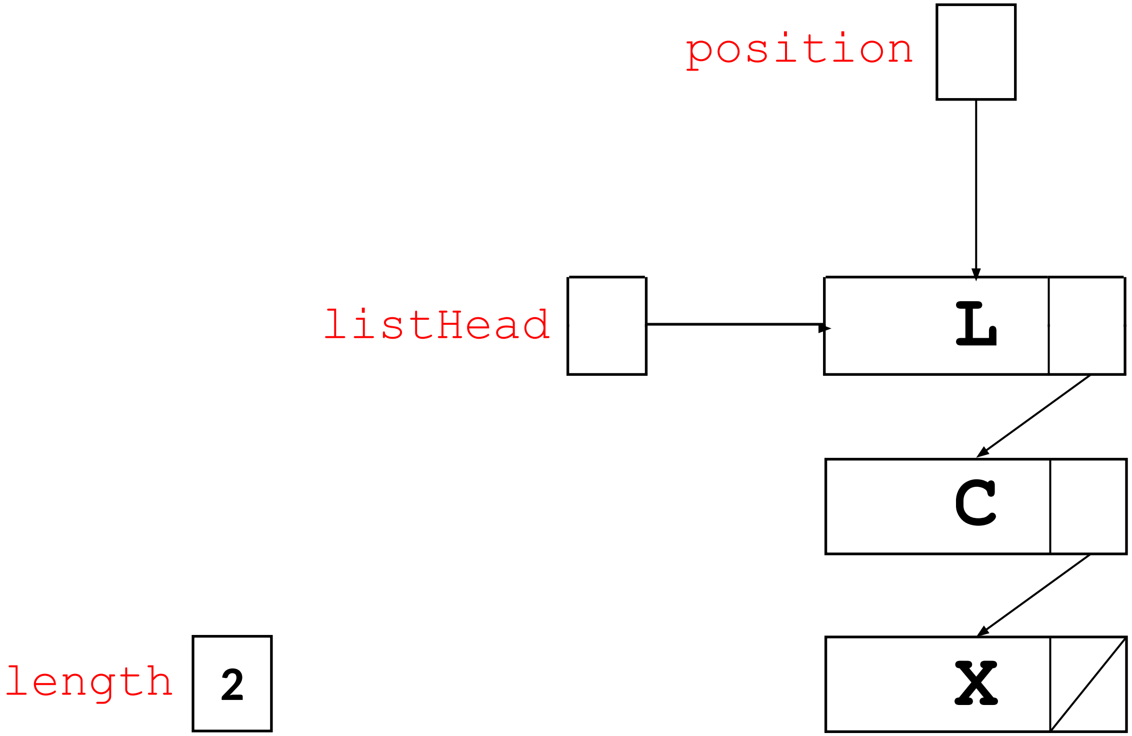
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('L')

unsortedlinkedlist.cpp

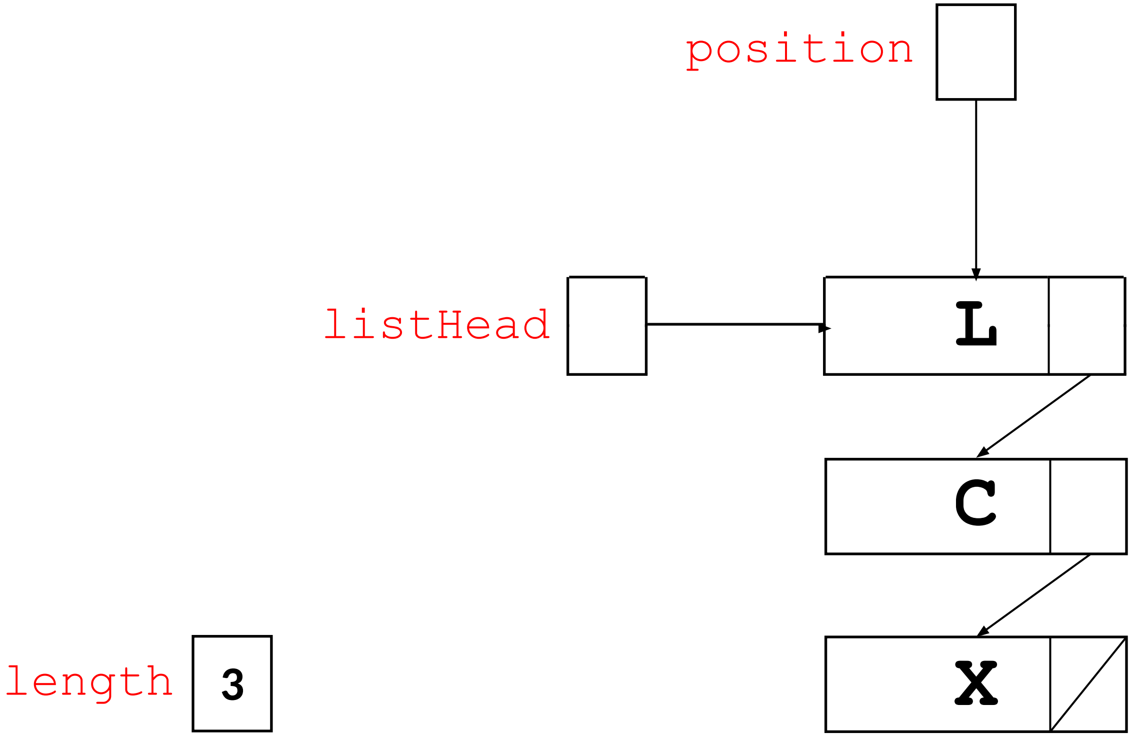
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('L')

unsortedlinkedlist.cpp

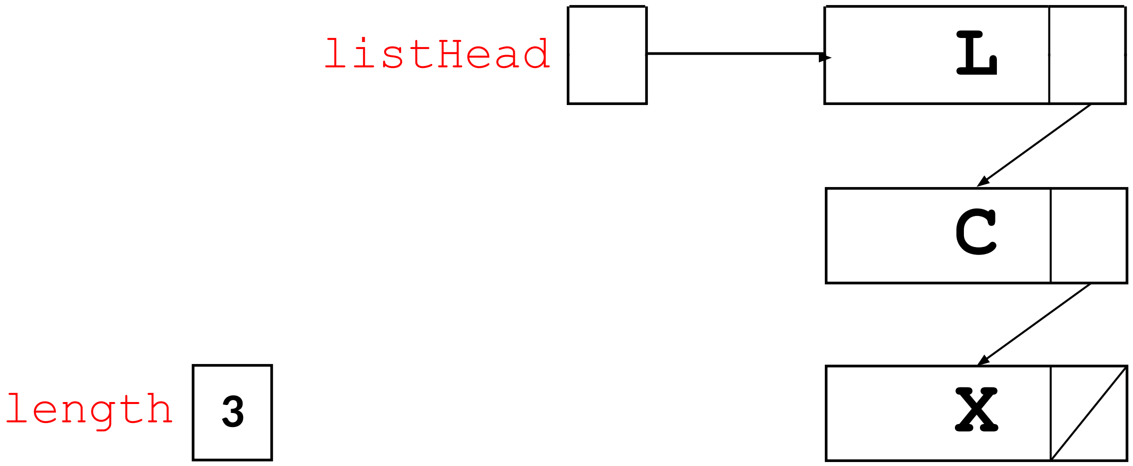
```
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* position;
  position = new NodeType;
  position->info = item;
  position->next = listHead;
  listHead = position;
  length++;
}
```



InsertItem('L')

unsortedlinkedlist.cpp

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



unsortedlinkedlist.cpp

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

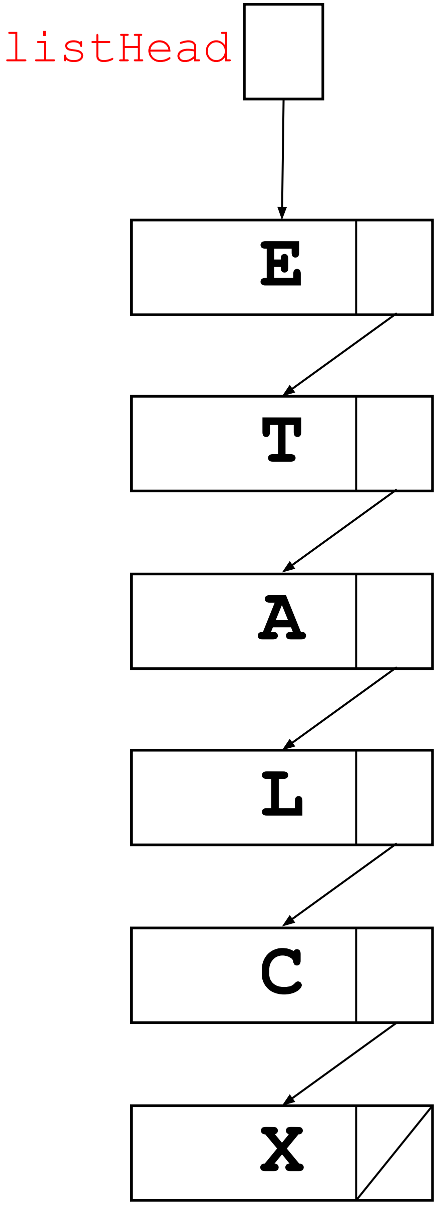
O(1)

unsortedlinkedlist.cpp

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

unsortedlinkedlist.cpp

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

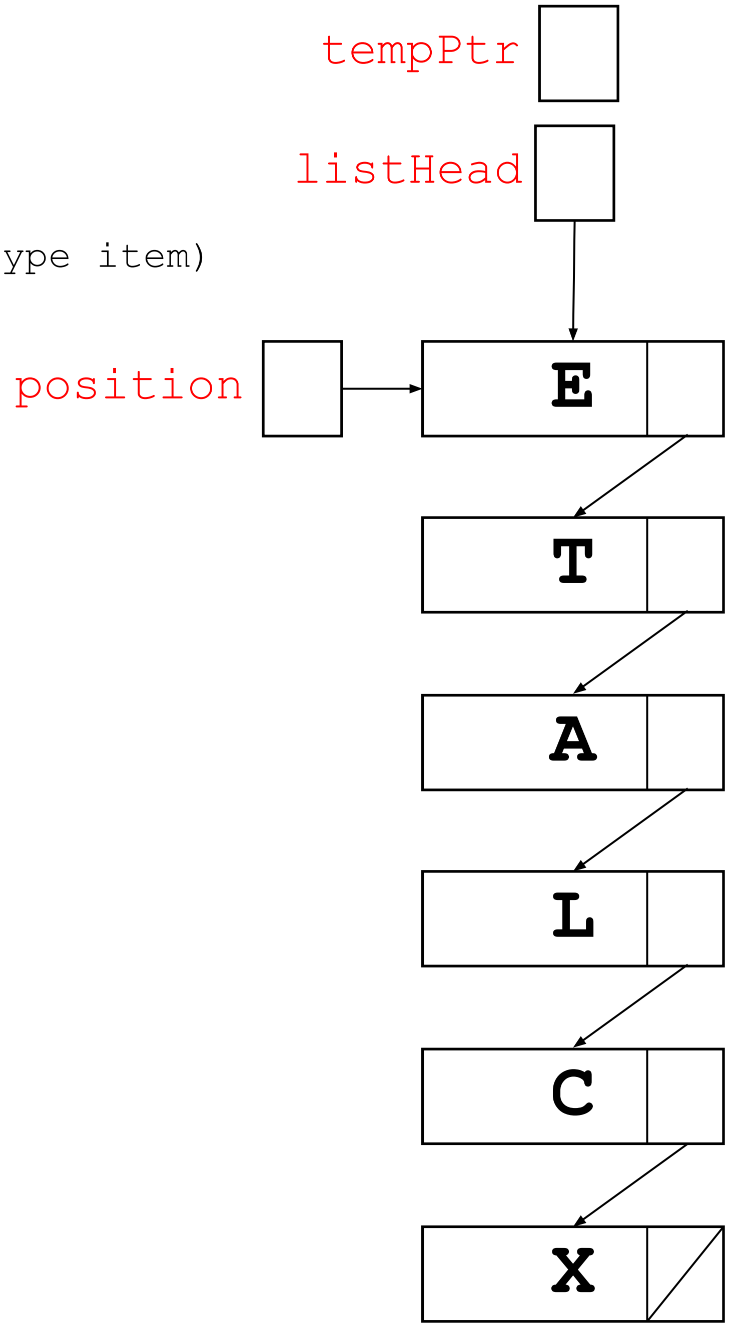


length 6

DeleteItem('L')

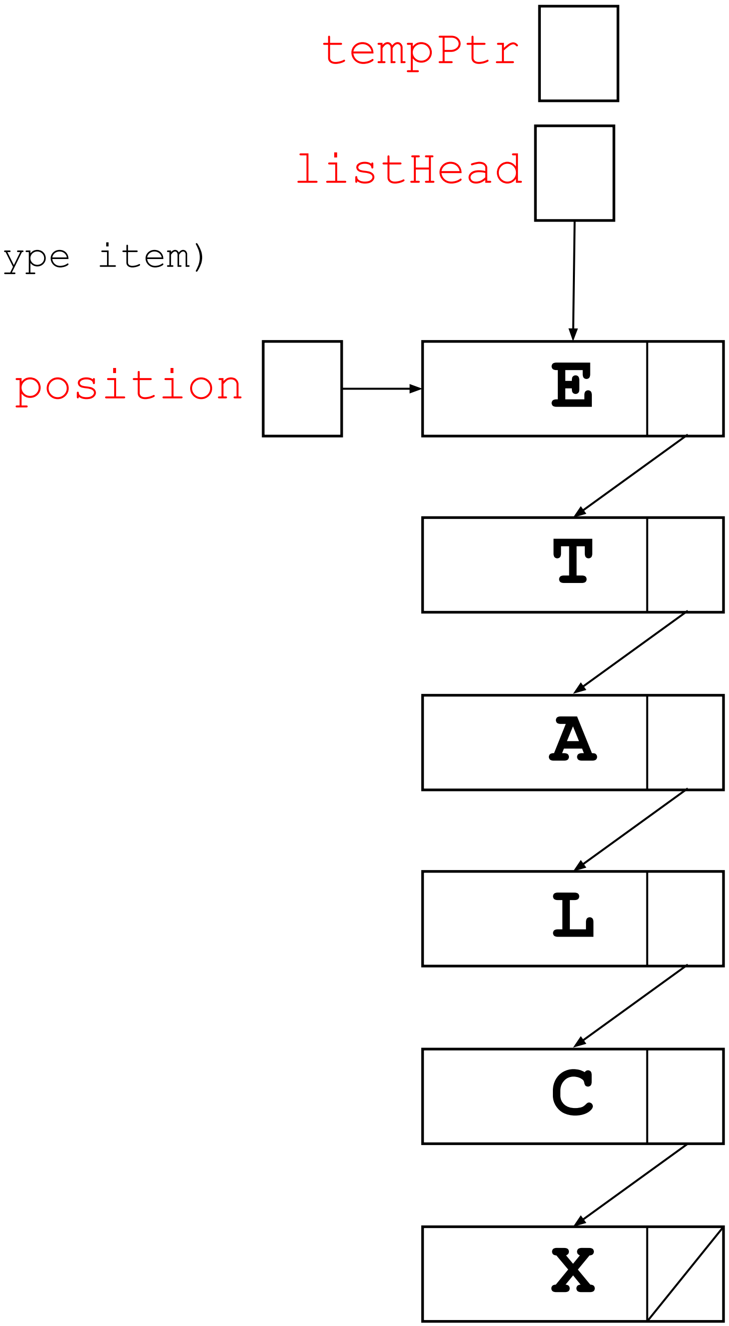
unsortedlinkedlist.cpp

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



DeleteItem('L')

unsortedlinkedlist.cpp

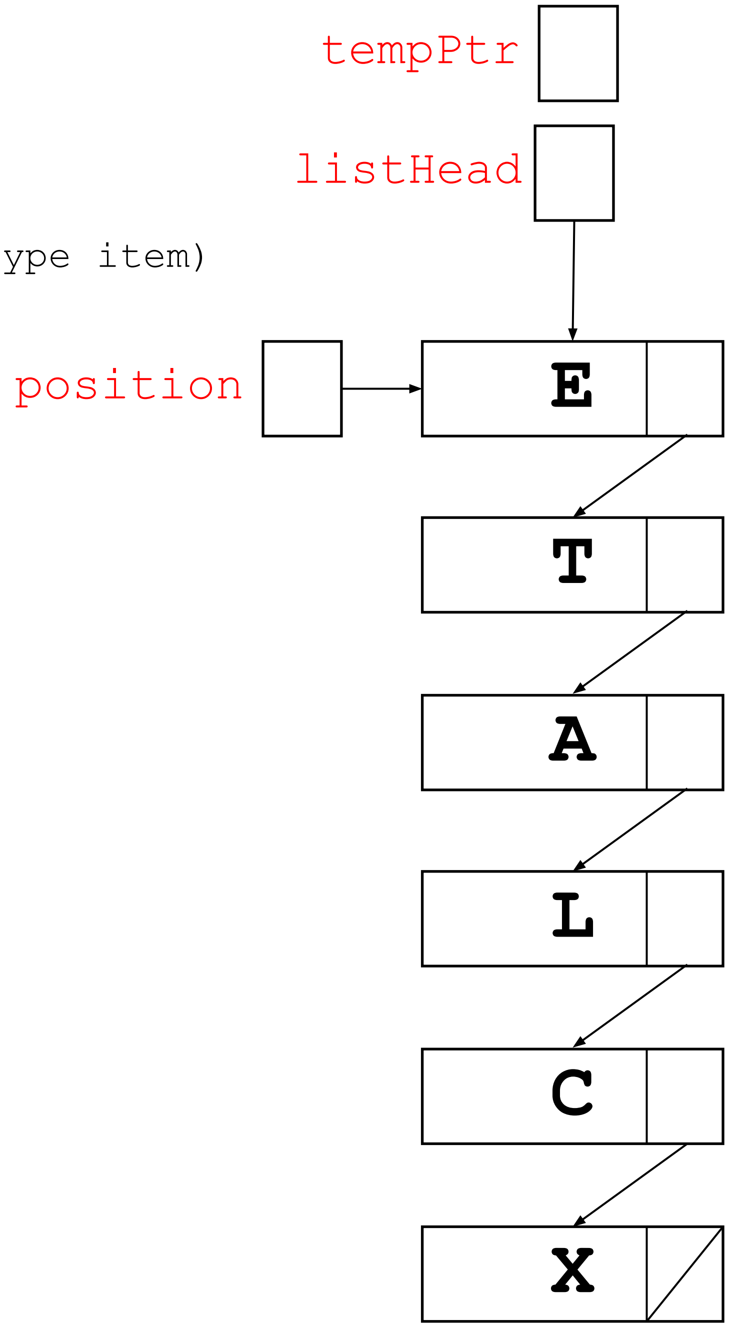


```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* position = listHead;
  NodeType* tempPtr;
  if (item == listHead->info)
  {
    tempPtr = position;
    listHead = listHead->next;
  }
  else
  {
    while (!(item==(position->next)->info))
      position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

DeleteItem('L')

unsortedlinkedlist.cpp

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

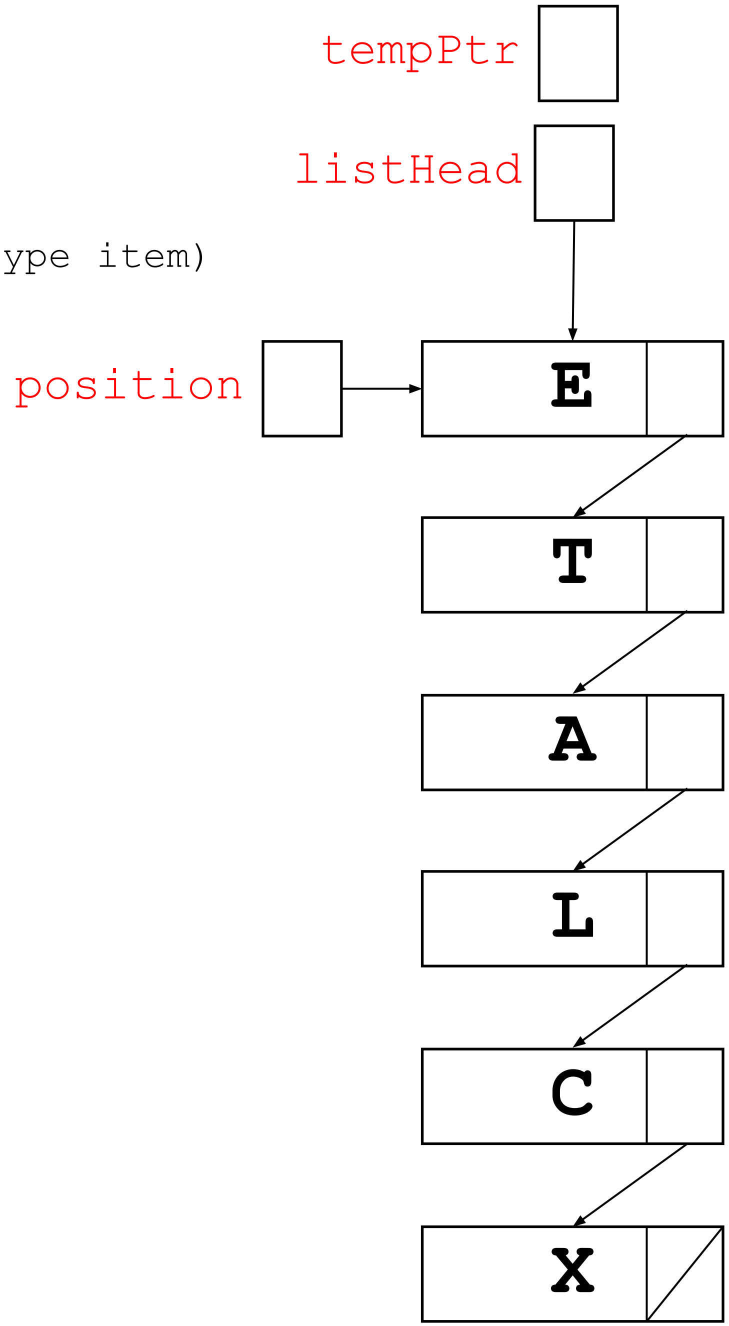


length 6

DeleteItem('L')

unsortedlinkedlist.cpp

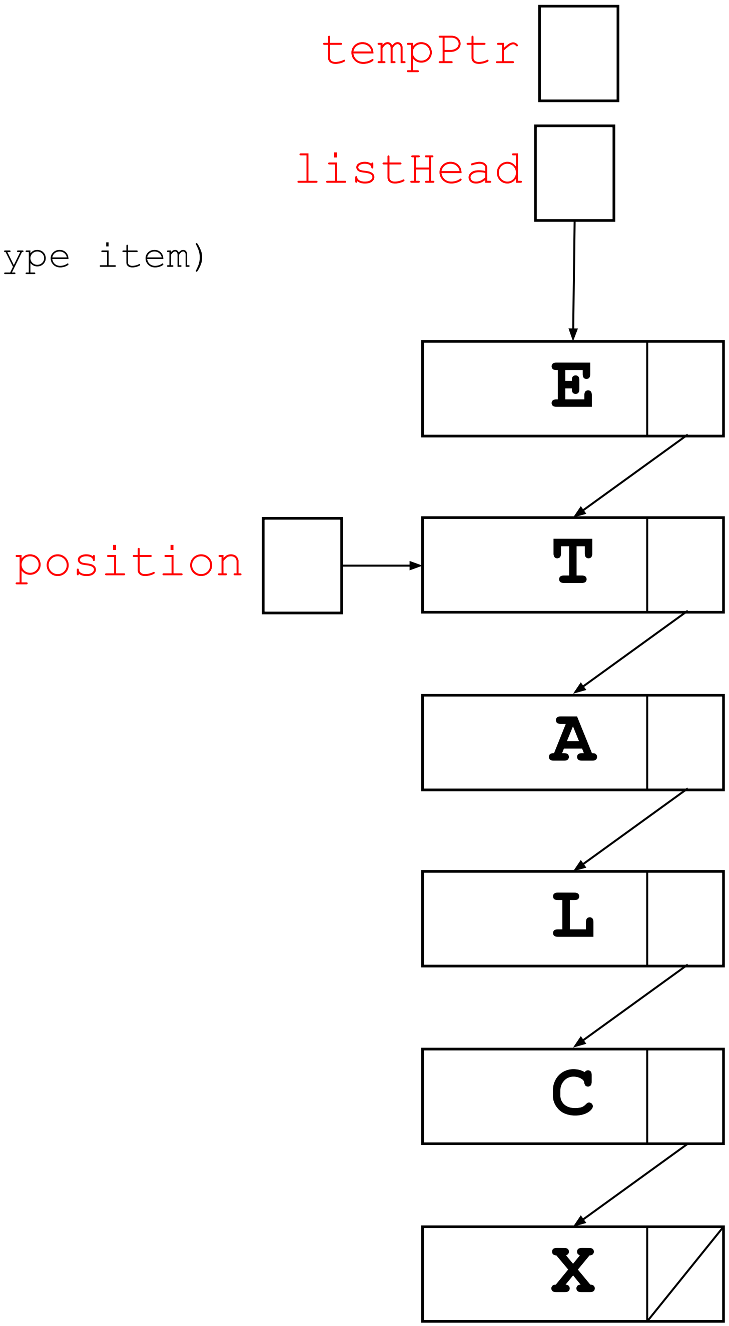
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* position = listHead;
  NodeType* tempPtr;
  if (item == listHead->info)
  {
    tempPtr = position;
    listHead = listHead->next;
  }
  else
  {
    while (!(item==(position->next)->info))
      position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



length 6

DeleteItem('L')

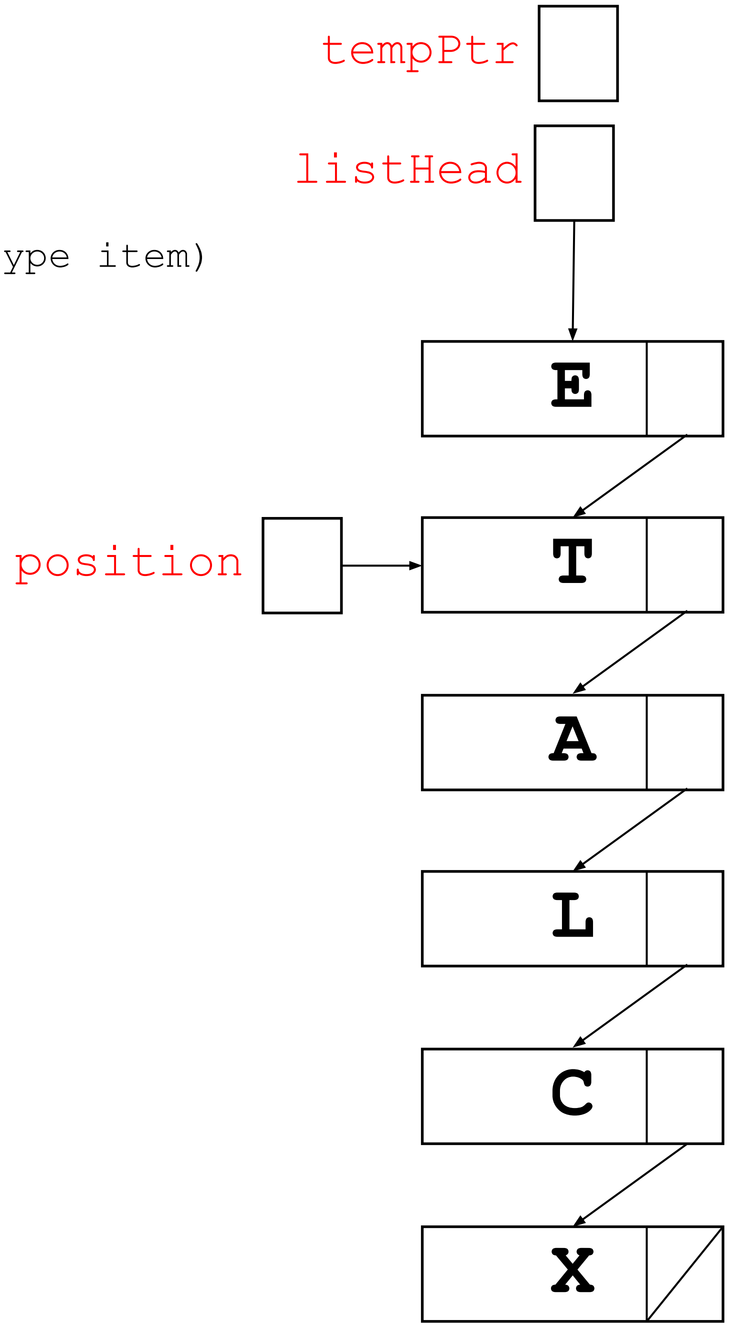
unsortedlinkedlist.cpp



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

DeleteItem('L')

unsortedlinkedlist.cpp

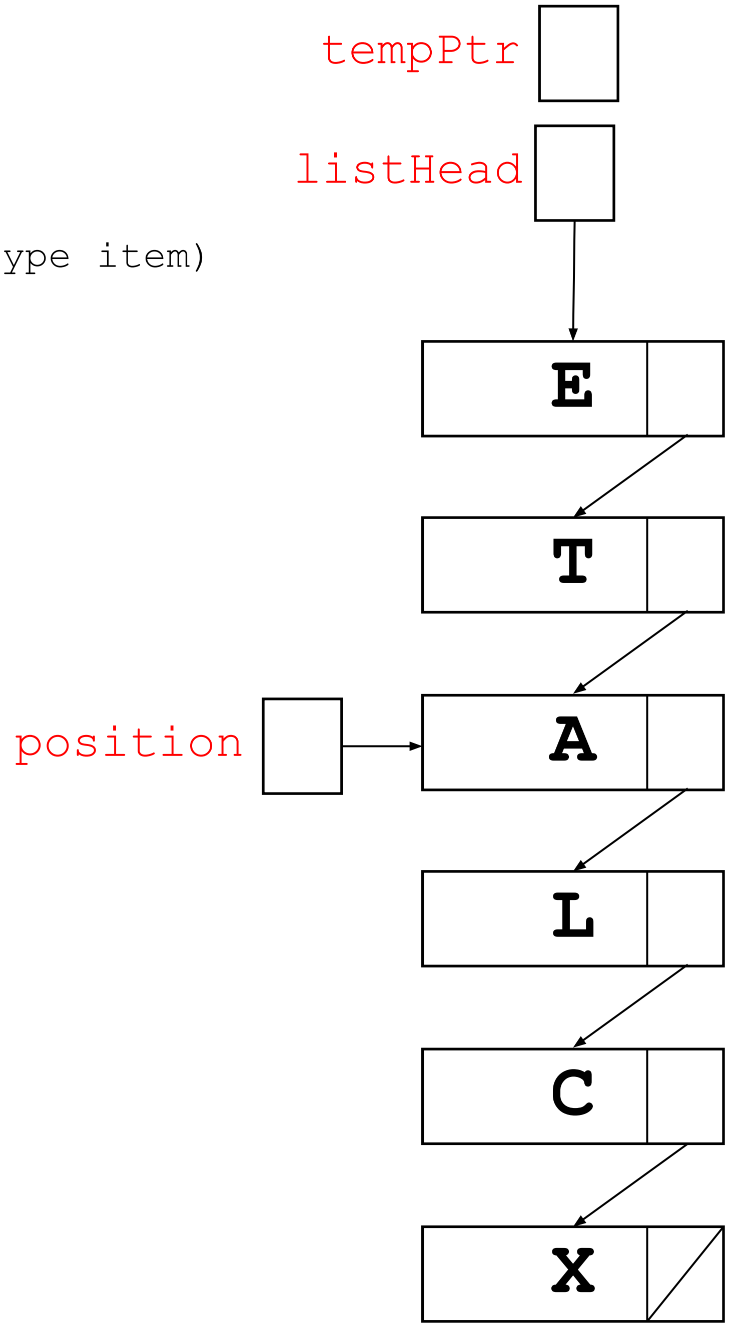


```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* position = listHead;
  NodeType* tempPtr;
  if (item == listHead->info)
  {
    tempPtr = position;
    listHead = listHead->next;
  }
  else
  {
    while (!(item==(position->next)->info))
      position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

DeleteItem('L')

unsortedlinkedlist.cpp

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

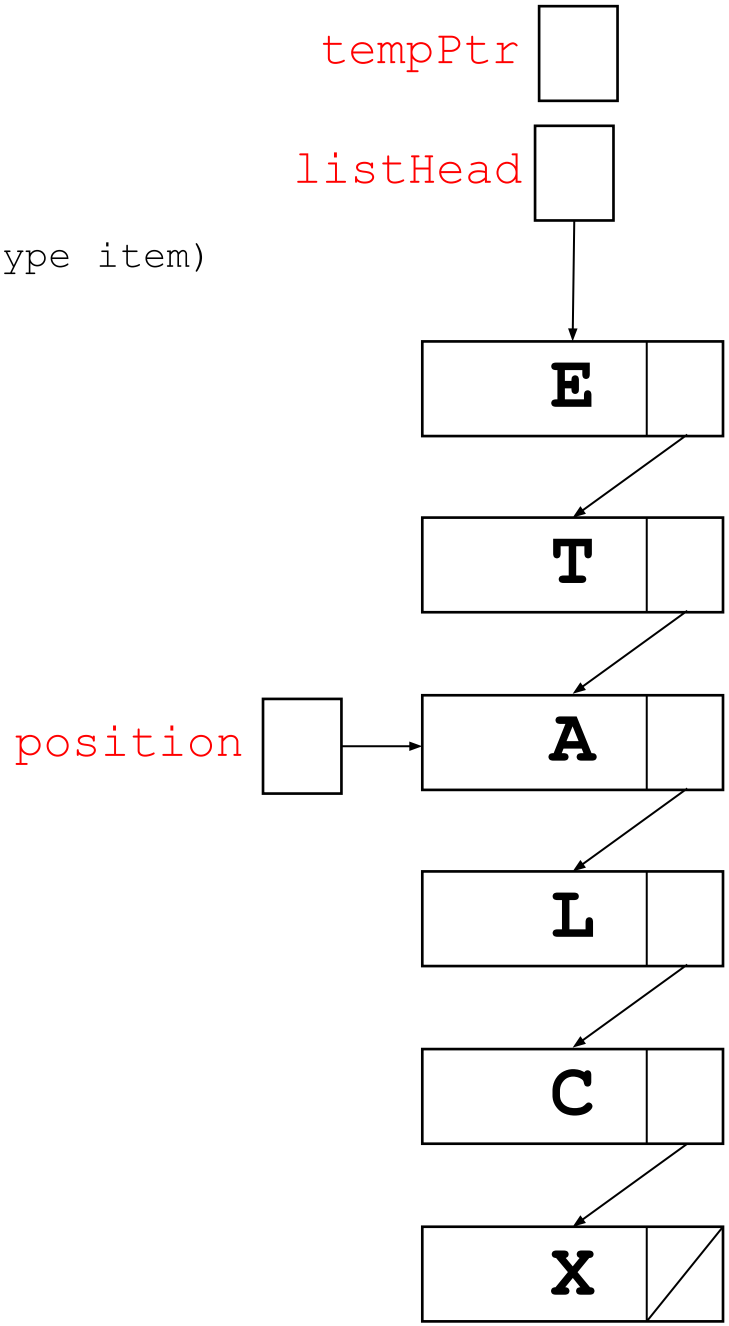


length 6

DeleteItem('L')

unsortedlinkedlist.cpp

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

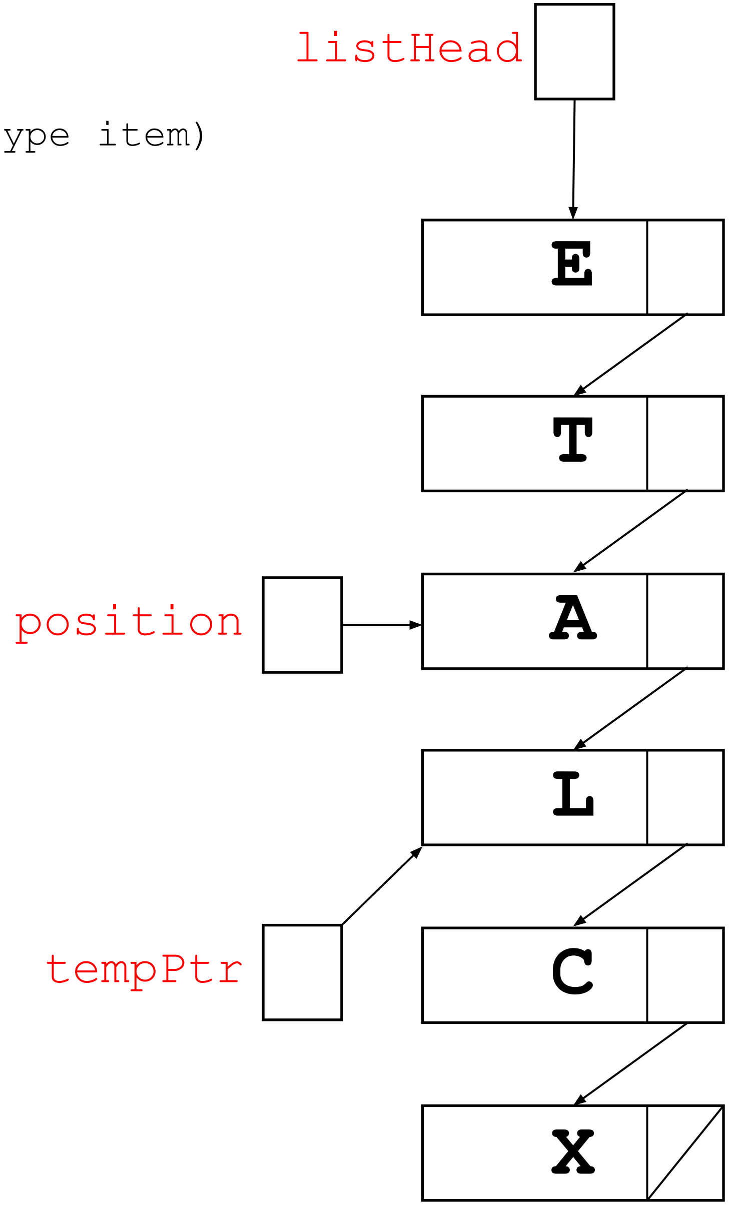


length 6

DeleteItem('L')

unsortedlinkedlist.cpp

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

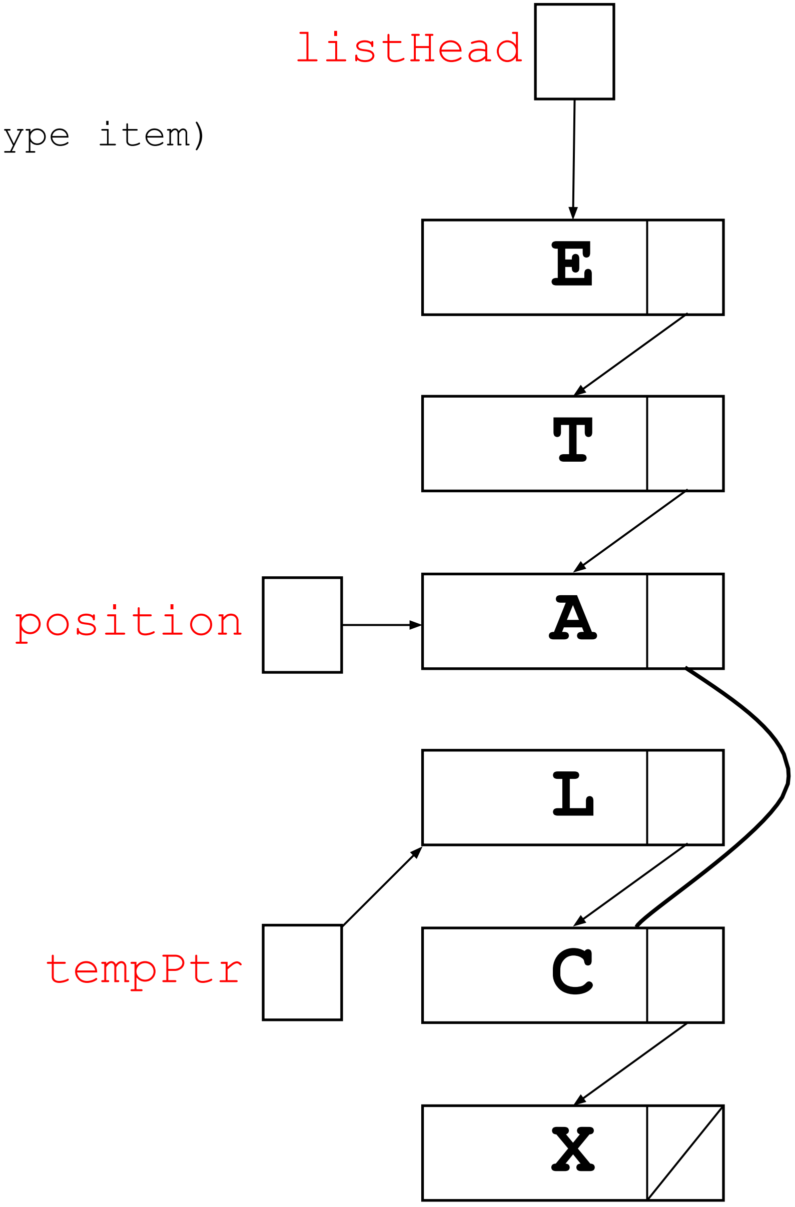


length **6**

DeleteItem('L')

unsortedlinkedlist.cpp

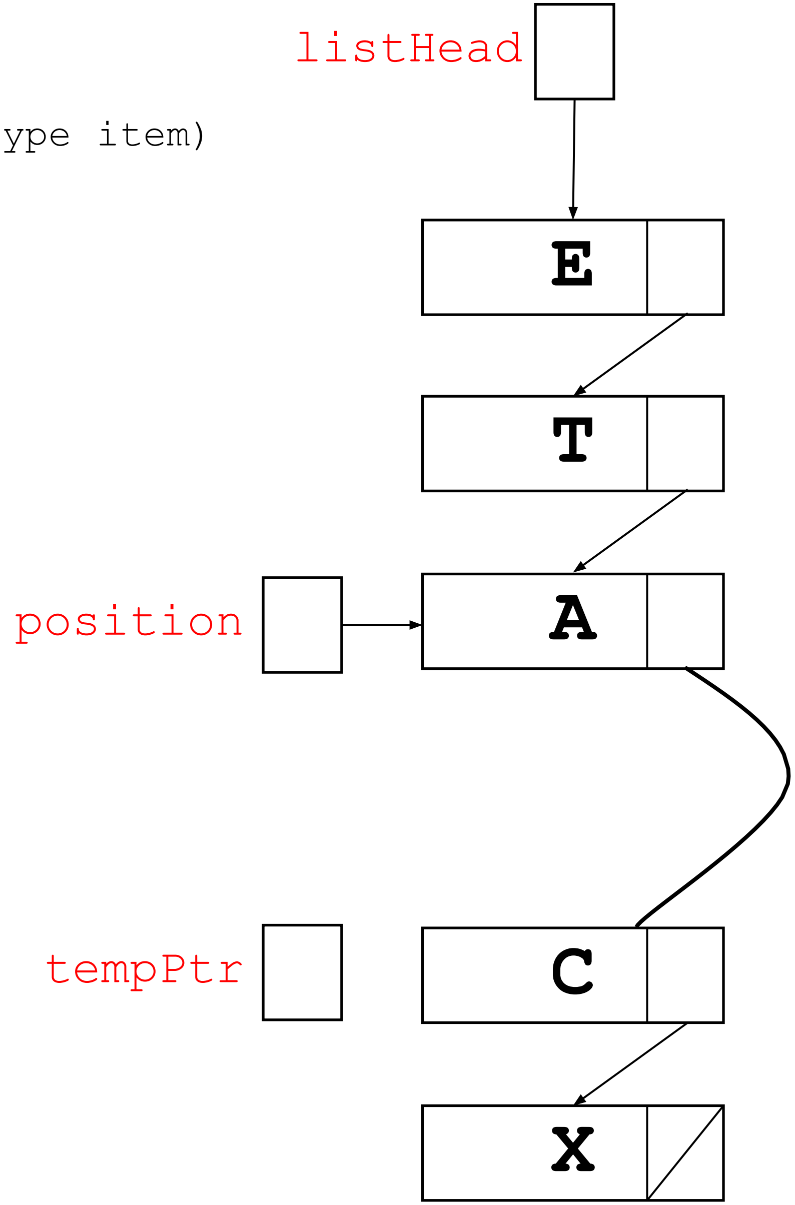
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* position = listHead;
  NodeType* tempPtr;
  if (item == listHead->info)
  {
    tempPtr = position;
    listHead = listHead->next;
  }
  else
  {
    while (!(item==(position->next)->info))
      position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



DeleteItem('L')

unsortedlinkedlist.cpp

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

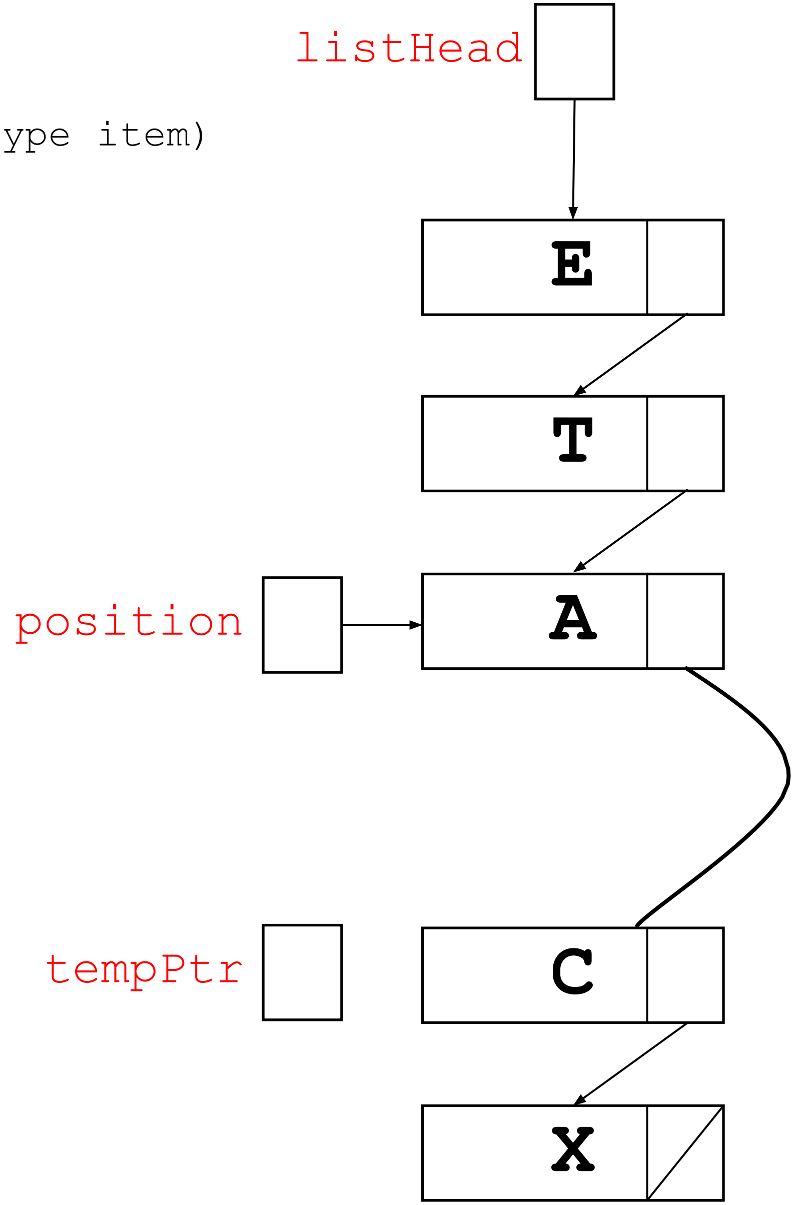


length 6

DeleteItem('L')

unsortedlinkedlist.cpp

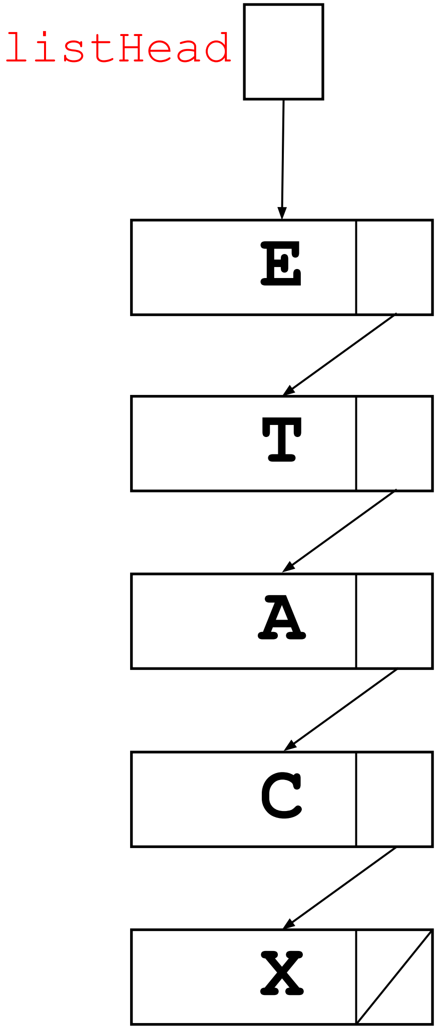
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* position = listHead;
  NodeType* tempPtr;
  if (item == listHead->info)
  {
    tempPtr = position;
    listHead = listHead->next;
  }
  else
  {
    while (!(item==(position->next)->info))
      position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



DeleteItem('L')

unsortedlinkedlist.cpp

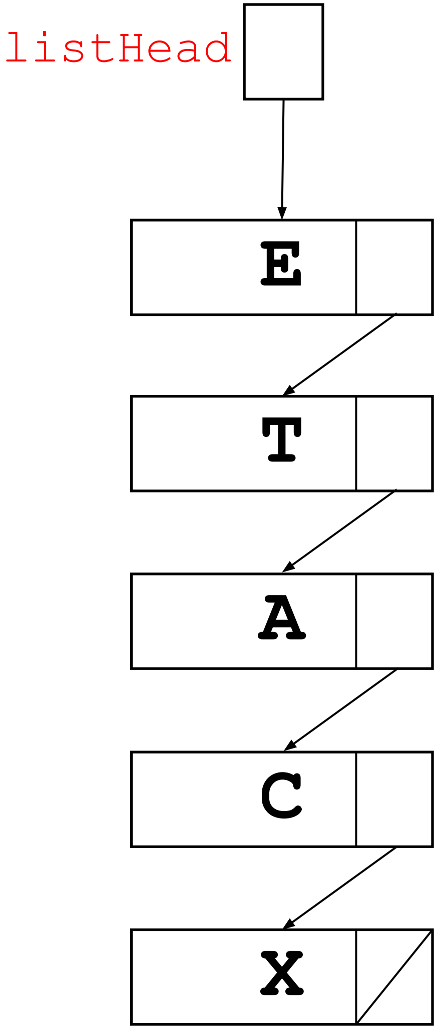
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* position = listHead;
  NodeType* tempPtr;
  if (item == listHead->info)
  {
    tempPtr = position;
    listHead = listHead->next;
  }
  else
  {
    while (!(item==(position->next)->info))
      position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



length 5

unsortedlinkedlist.cpp

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

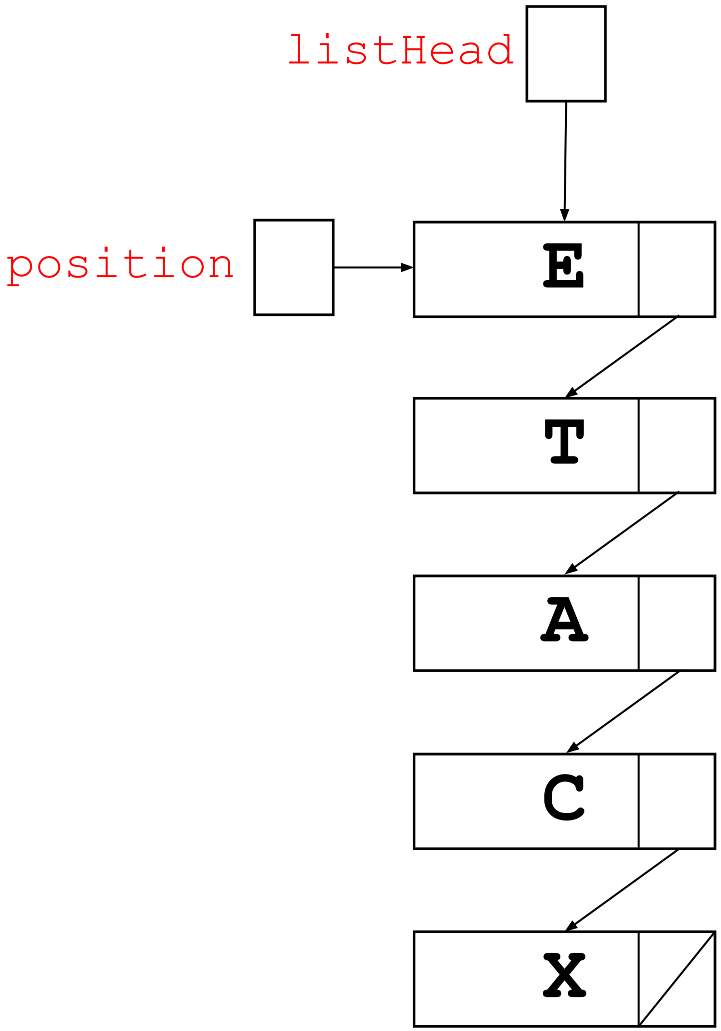


length 5

DeleteItem('E')

unsortedlinkedlist.cpp

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

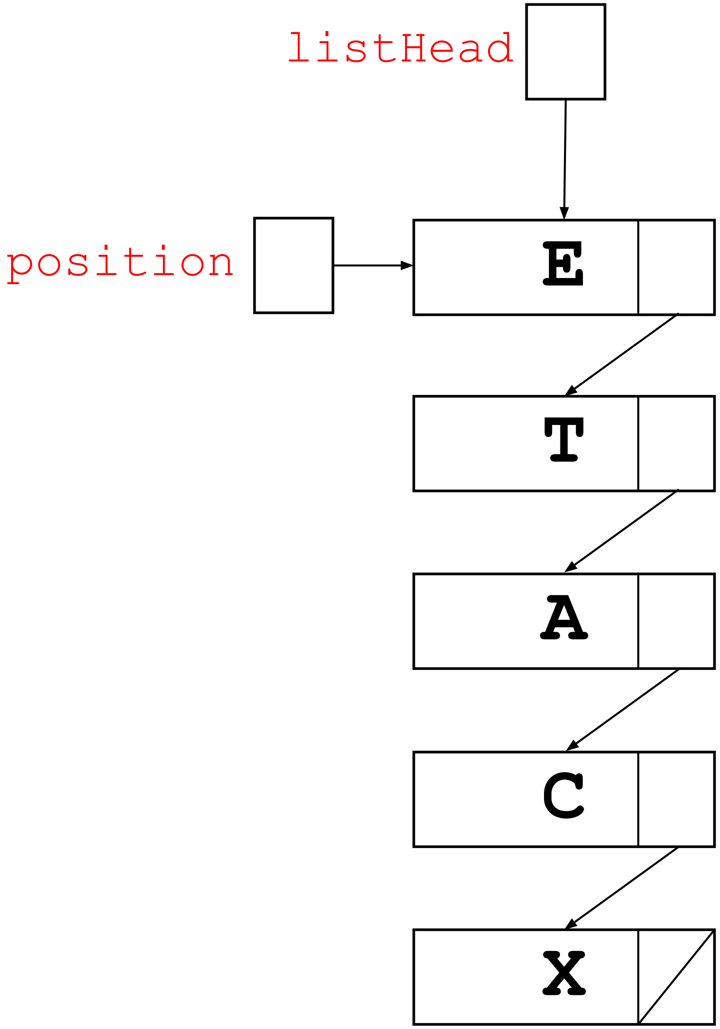


DeleteItem('E')

unsortedlinkedlist.cpp

tempPtr

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

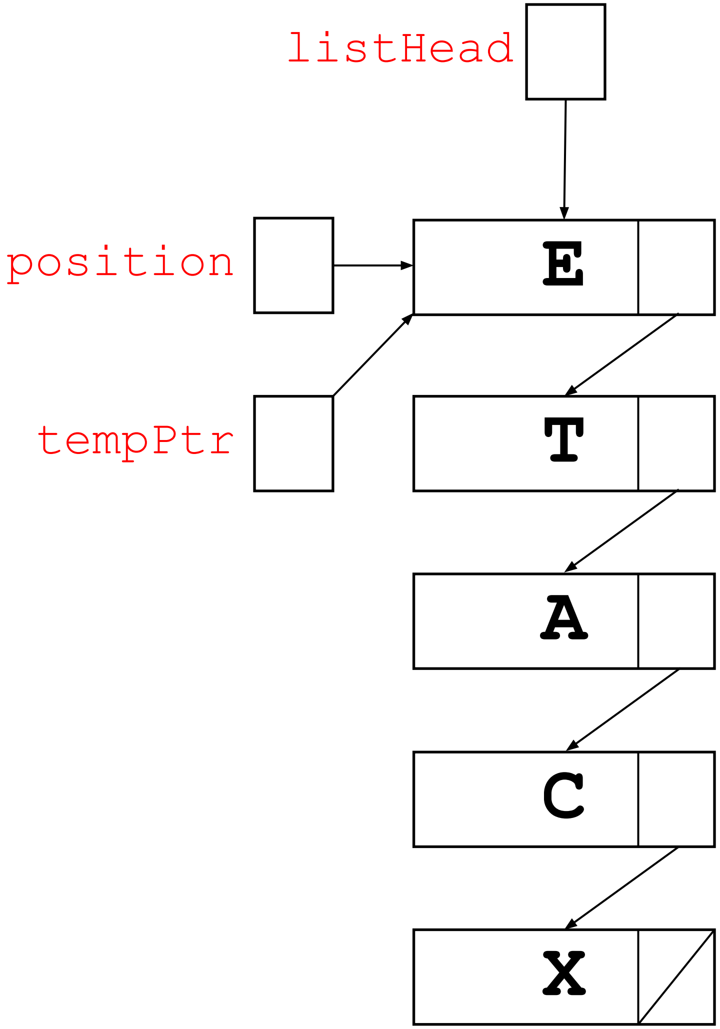


length 5

DeleteItem('E')

unsortedlinkedlist.cpp

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

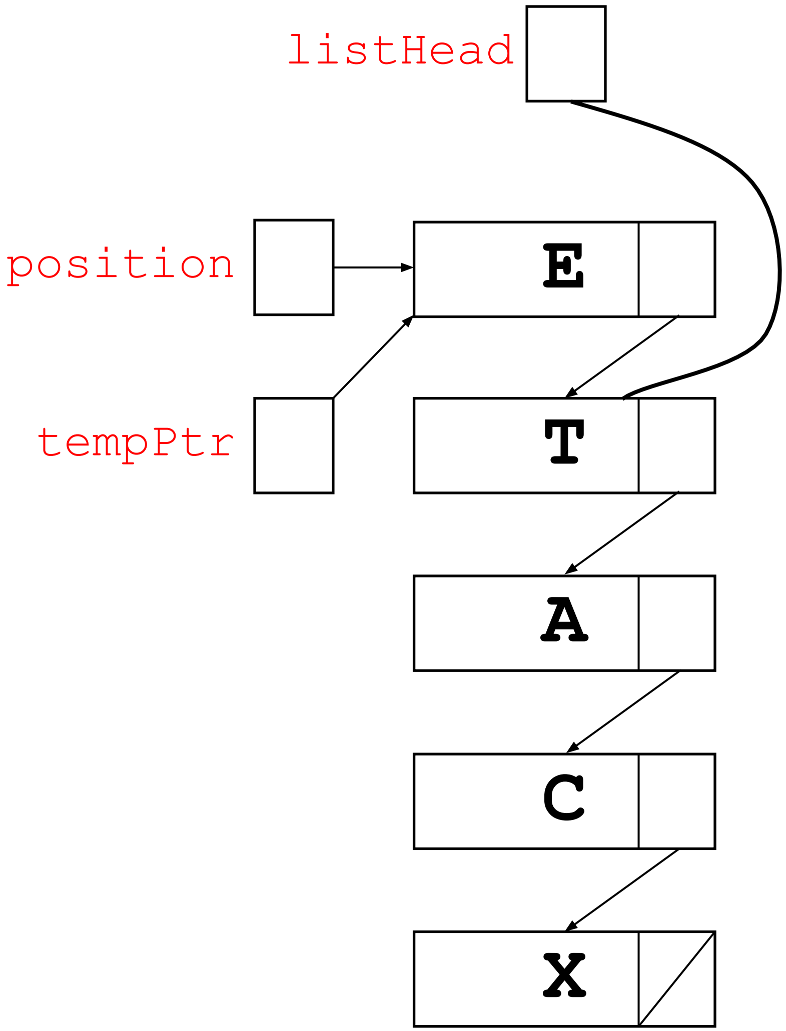


length 5

DeleteItem('E')

unsortedlinkedlist.cpp

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

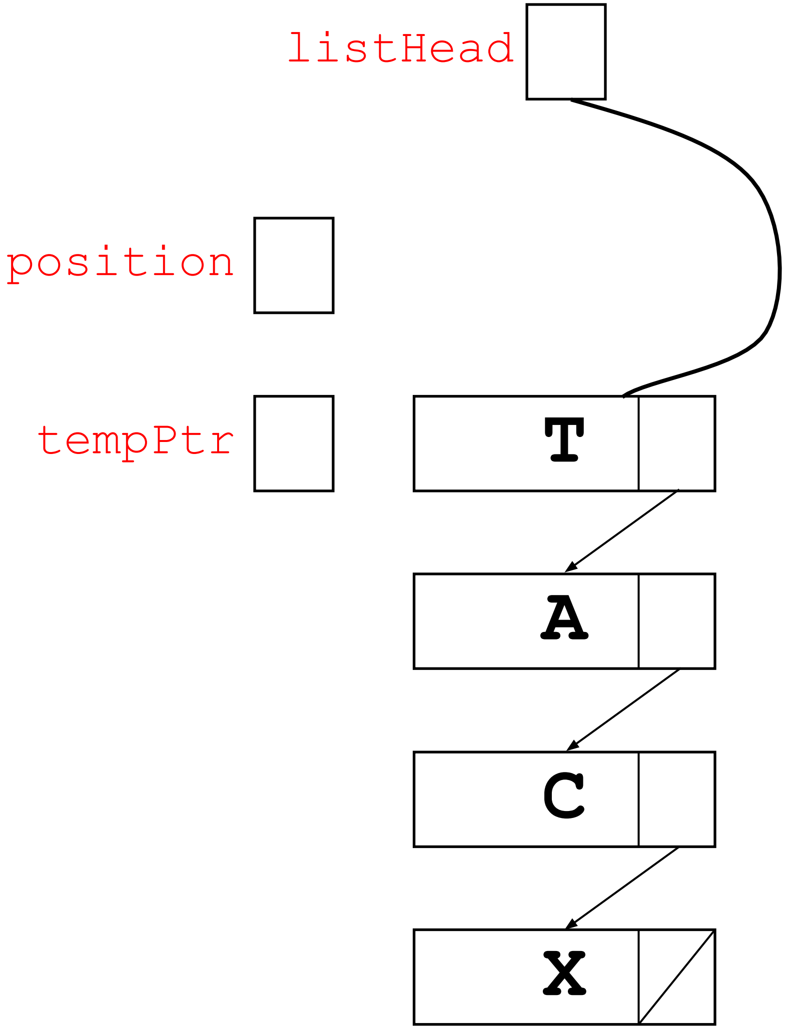


length 5

DeleteItem('E')

unsortedlinkedlist.cpp

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

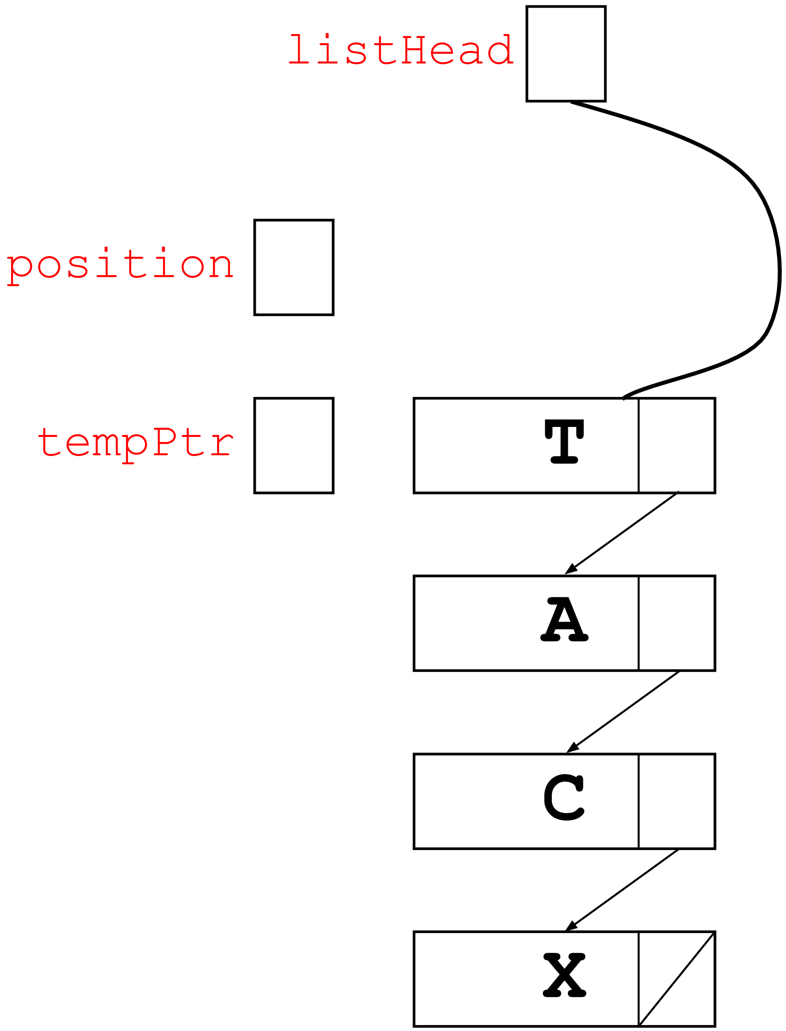


length **5**

DeleteItem('E')

unsortedlinkedlist.cpp

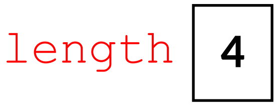
```
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* position = listHead;
  NodeType* tempPtr;
  if (item == listHead->info)
  {
    tempPtr = position;
    listHead = listHead->next;
  }
  else
  {
    while (!(item==(position->next)->info))
      position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

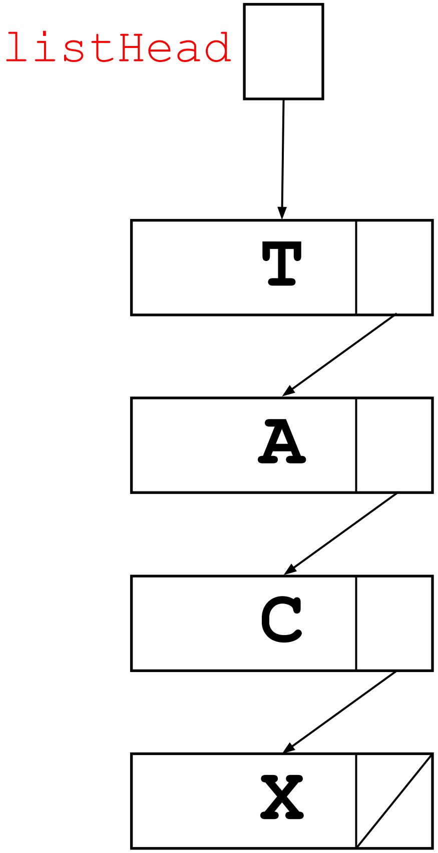


DeleteItem('E')

unsortedlinkedlist.cpp

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





unsortedlinkedlist.cpp

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

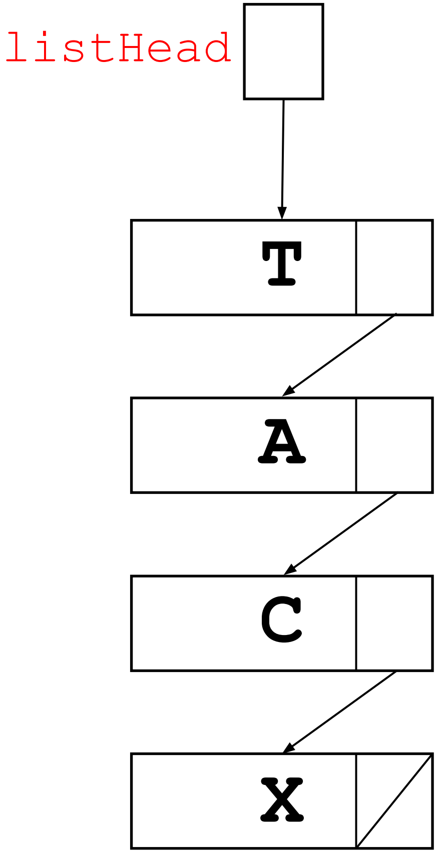
O(N)

unsortedlinkedlist.cpp

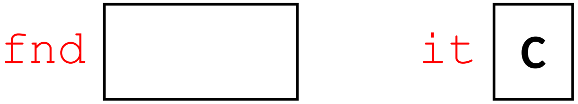
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* position = listHead;
  bool moreToSearch = (position != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == position->info)
    {
      found = true;
    }
    else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

unsortedlinkedlist.cpp

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

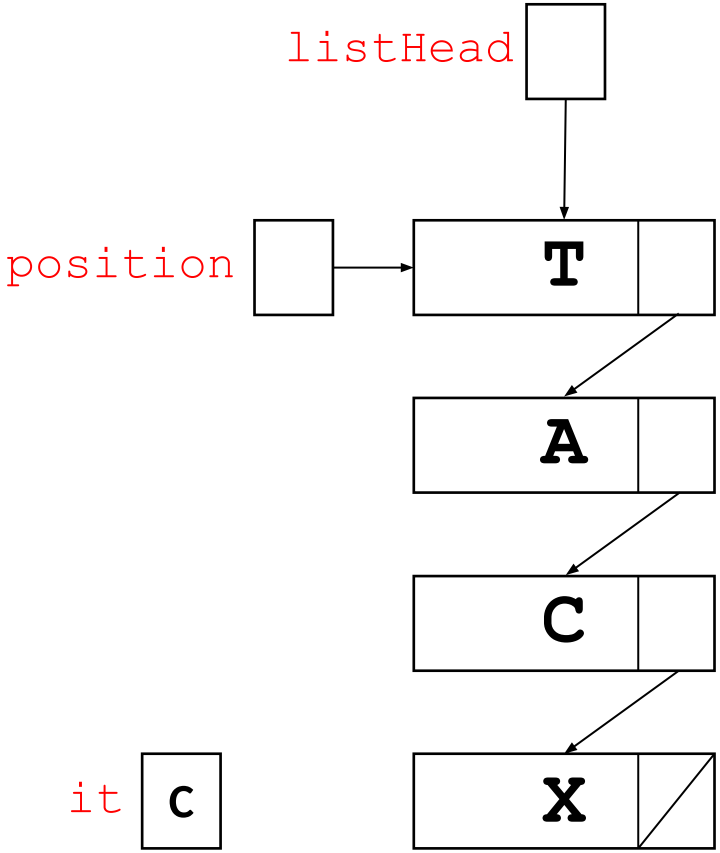


RetrieveItem(it,fnd)



unsortedlinkedlist.cpp

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

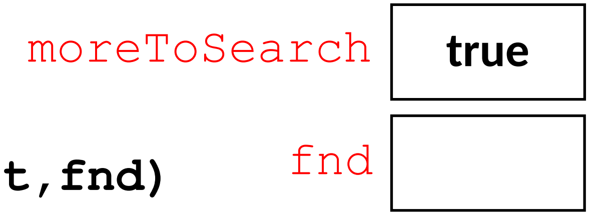


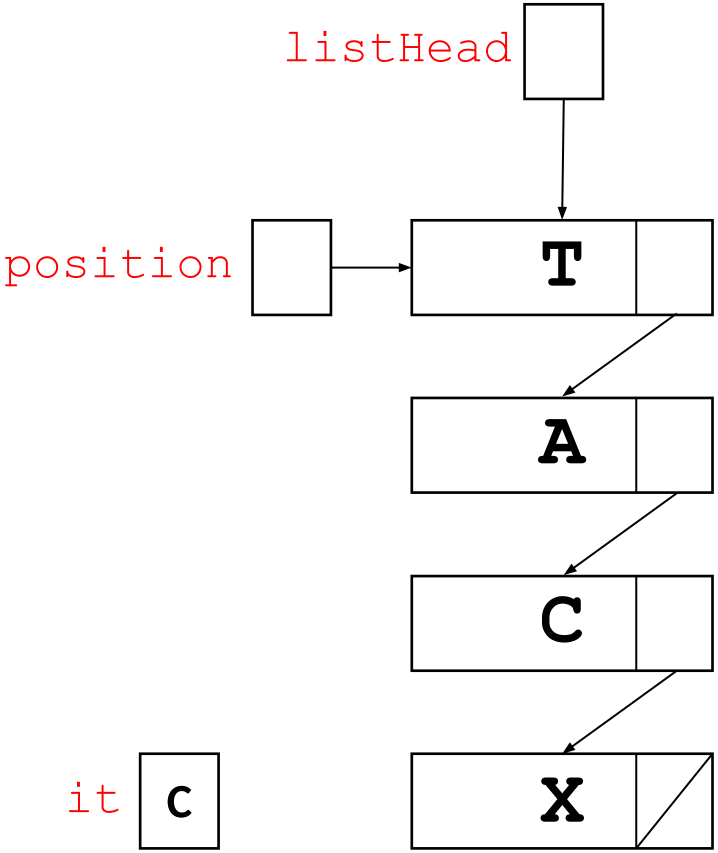
RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

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

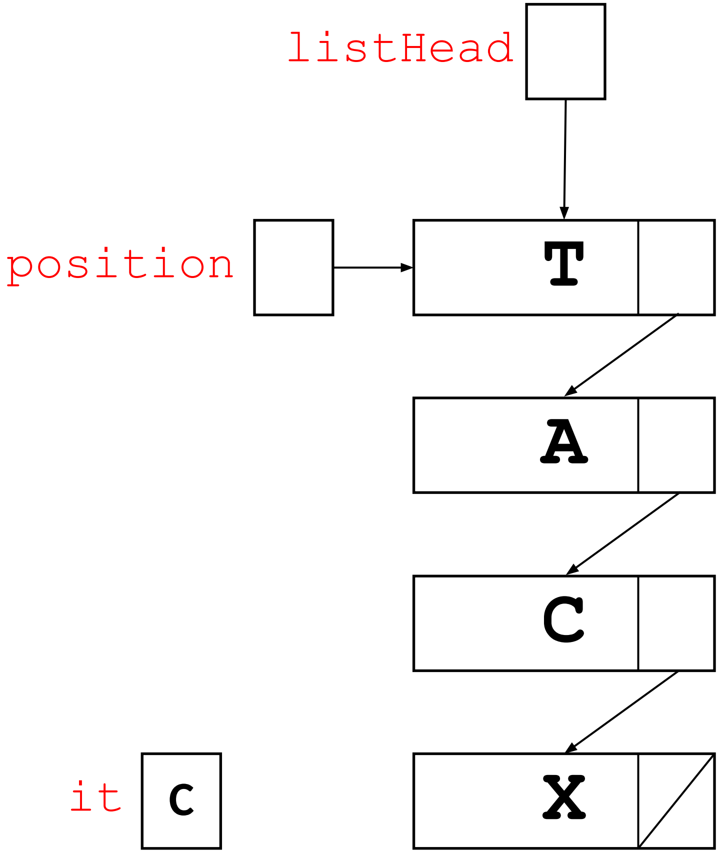
RetrieveItem(it,fnd)

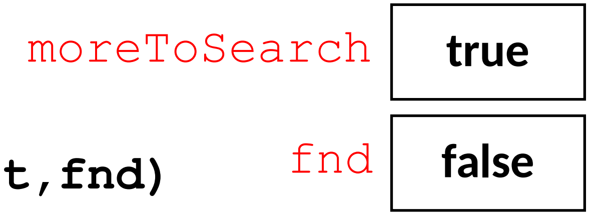




unsortedlinkedlist.cpp

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

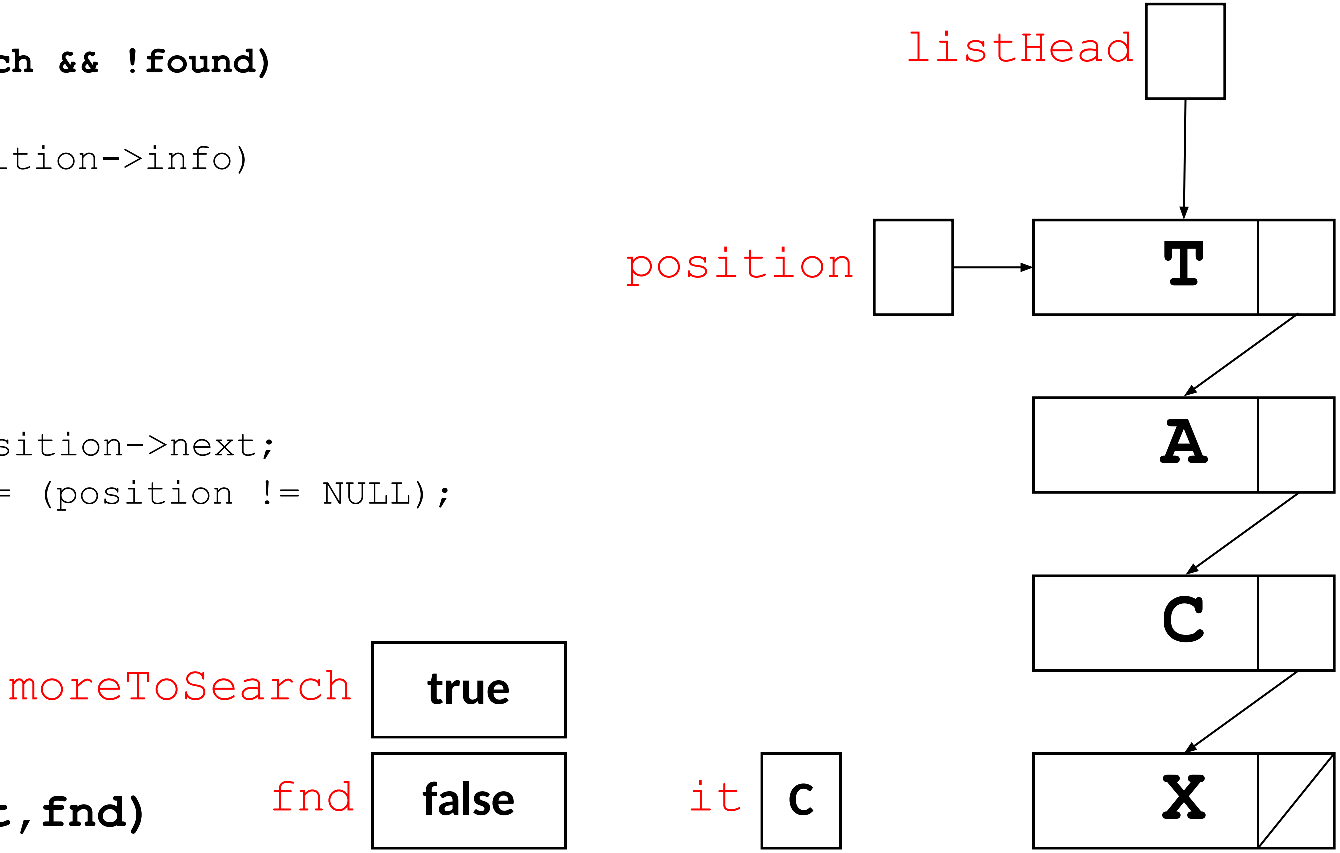




RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

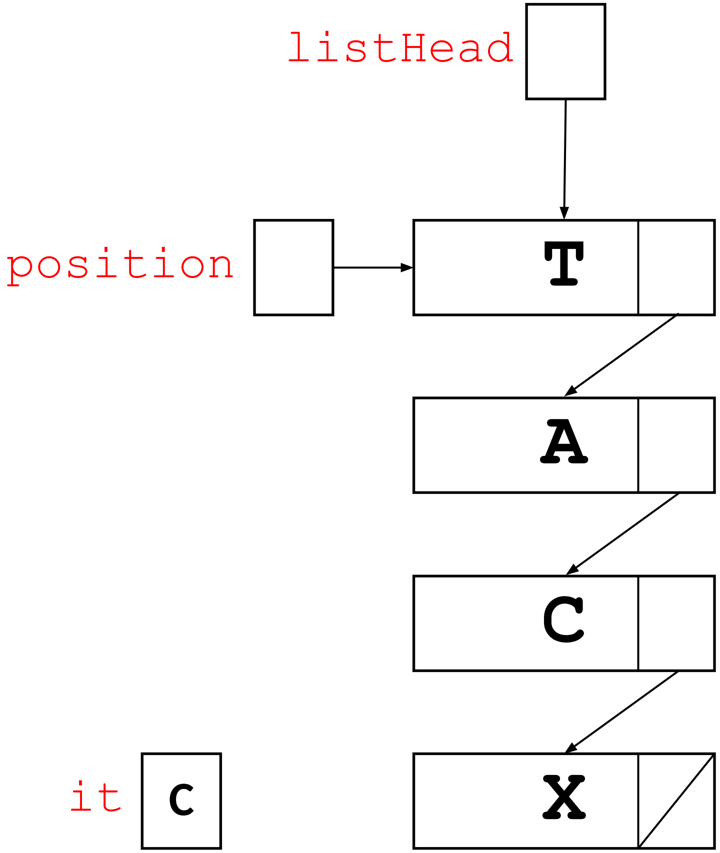
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* position = listHead;
  bool moreToSearch = (position != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == position->info)
    {
      found = true;
    }
    else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

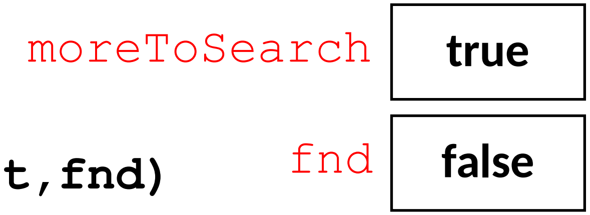


RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

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

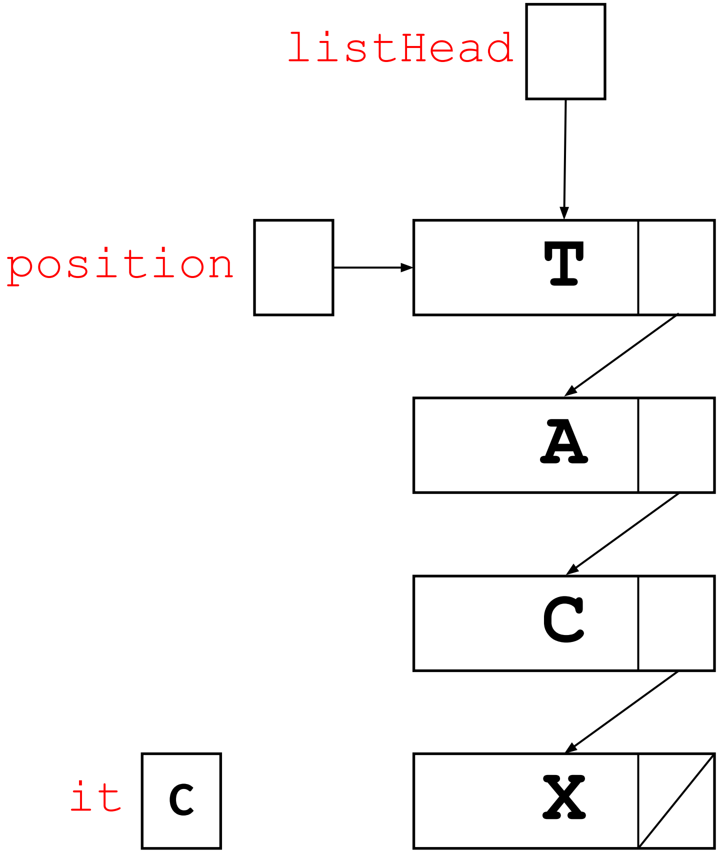


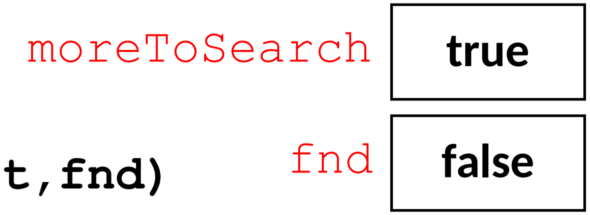


RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

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

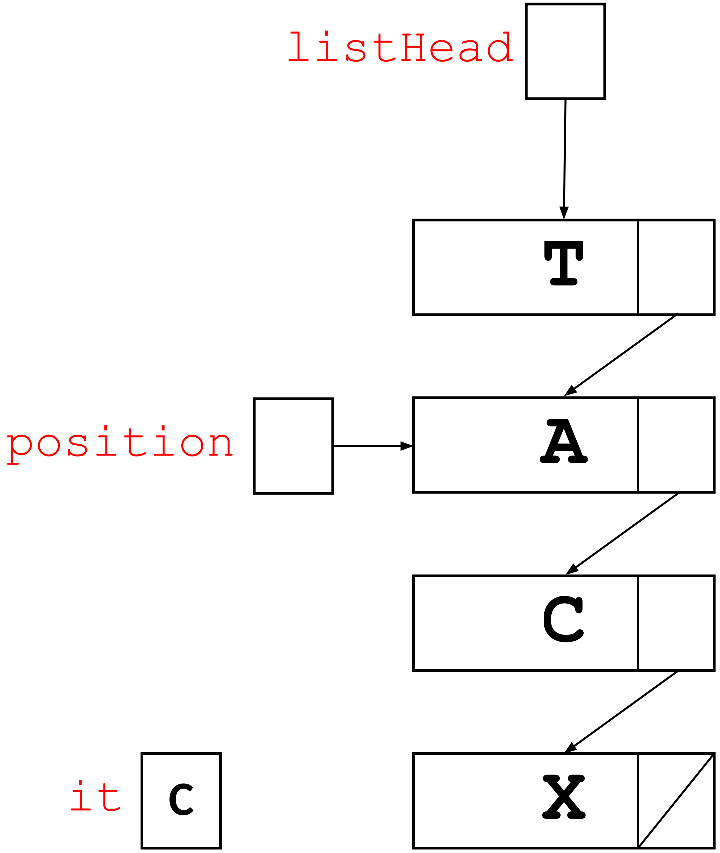




RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

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



moreToSearch **true**

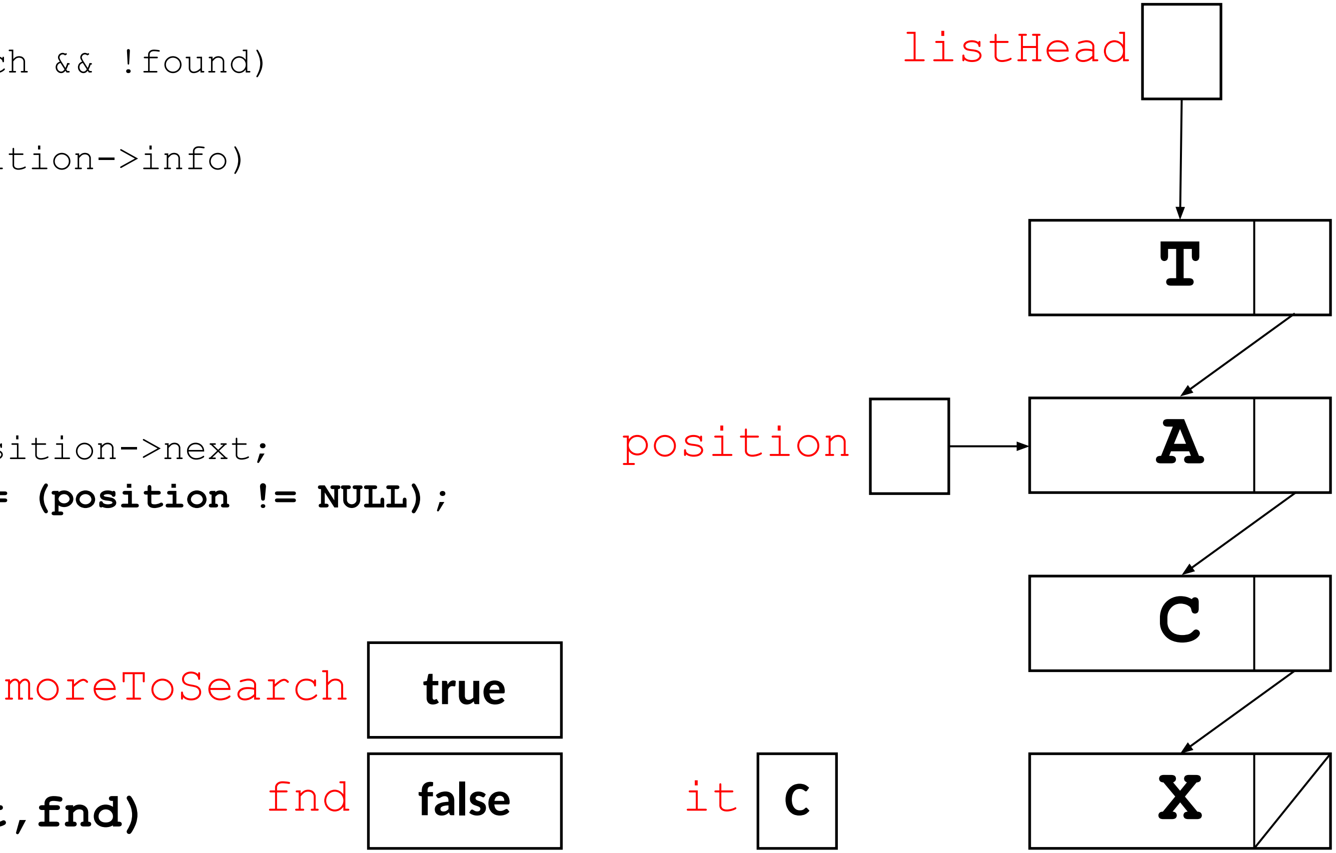
fnd **false**

RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

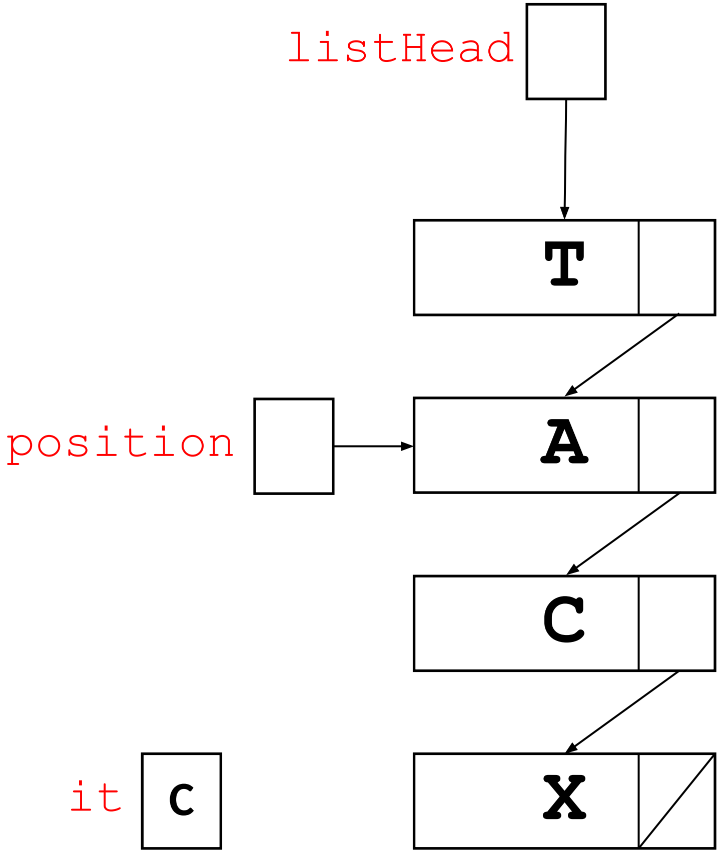
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* position = listHead;
  bool moreToSearch = (position != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == position->info)
    {
      found = true;
    }
    else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

RetrieveItem(it,fnd)



unsortedlinkedlist.cpp

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

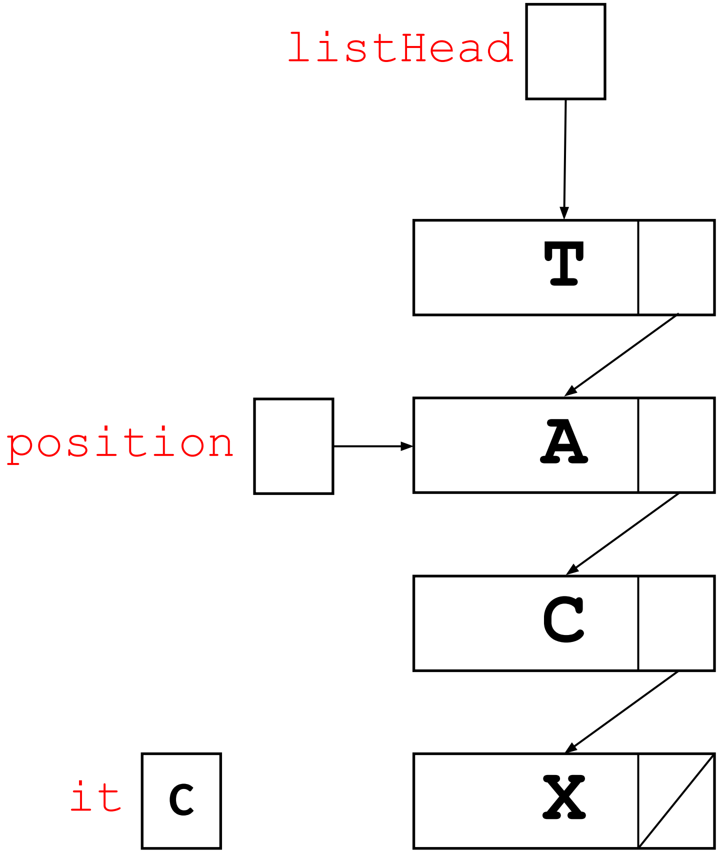


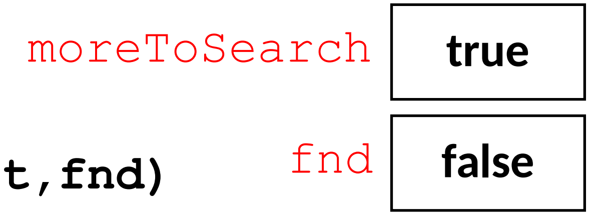
moreToSearch **true**

RetrieveItem(it,fnd) fnd **false**

unsortedlinkedlist.cpp

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

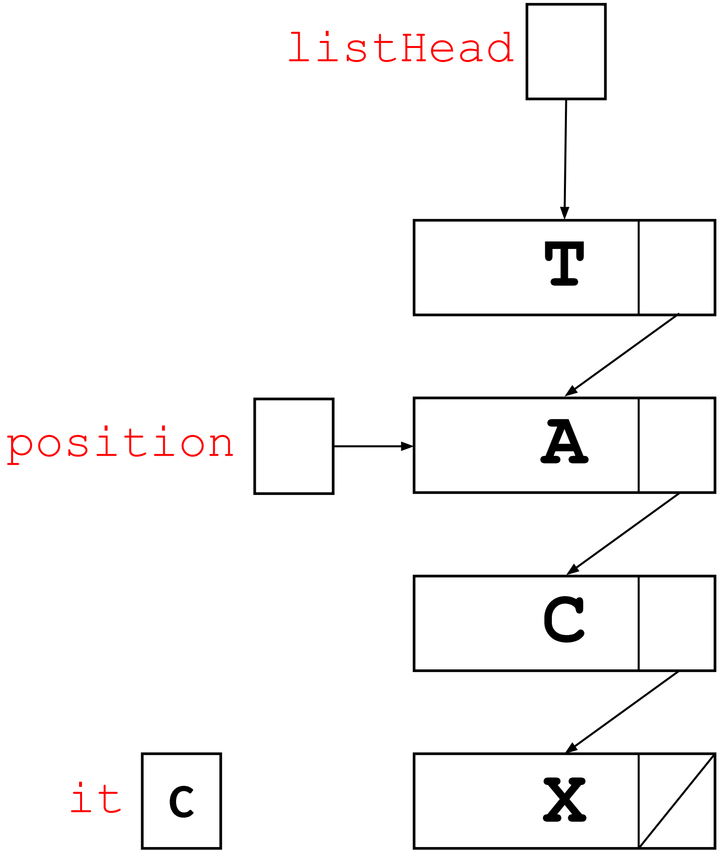


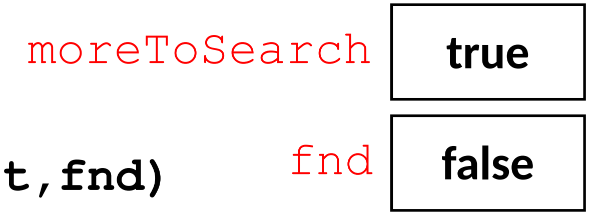


RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

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

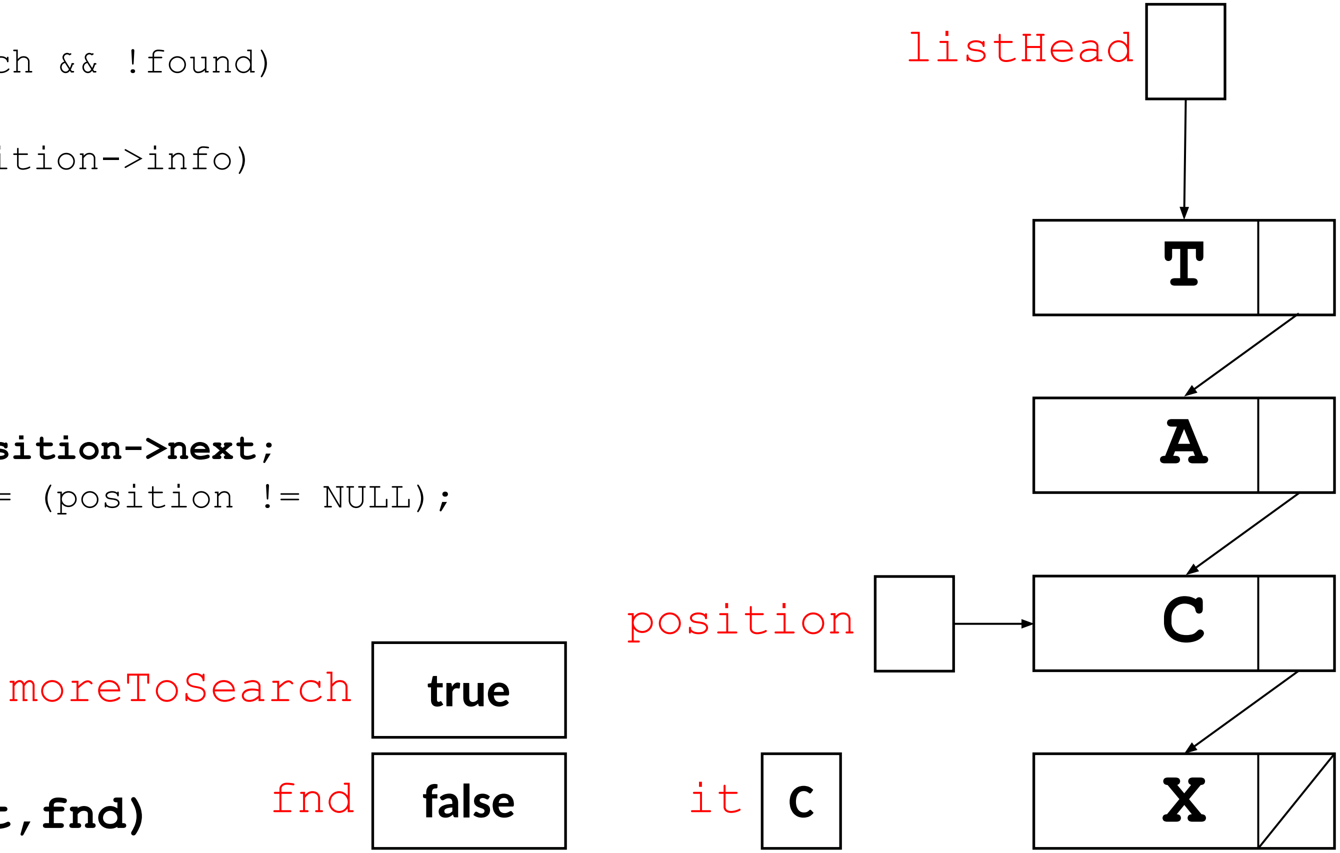




RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

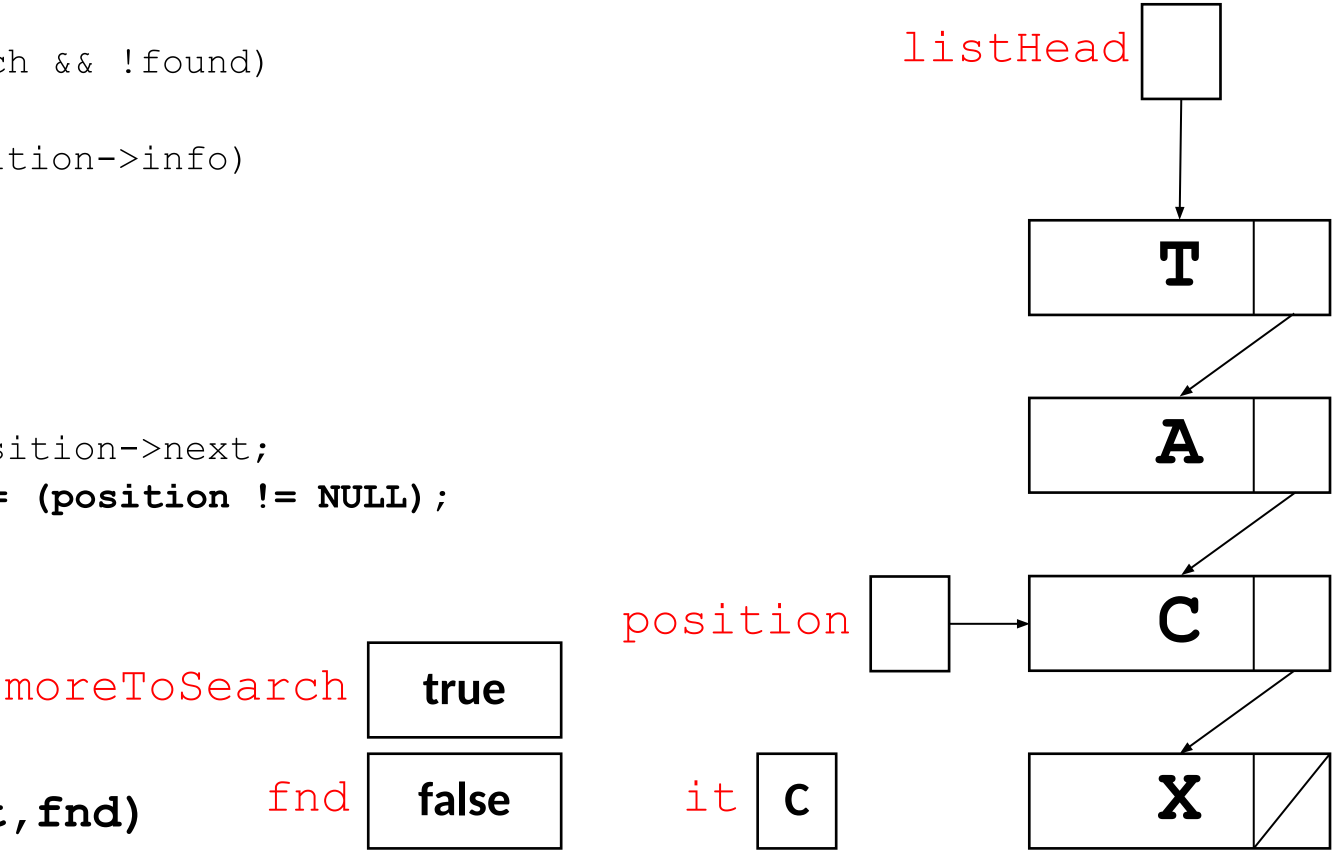
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* position = listHead;
  bool moreToSearch = (position != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == position->info)
    {
      found = true;
    }
    else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



RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

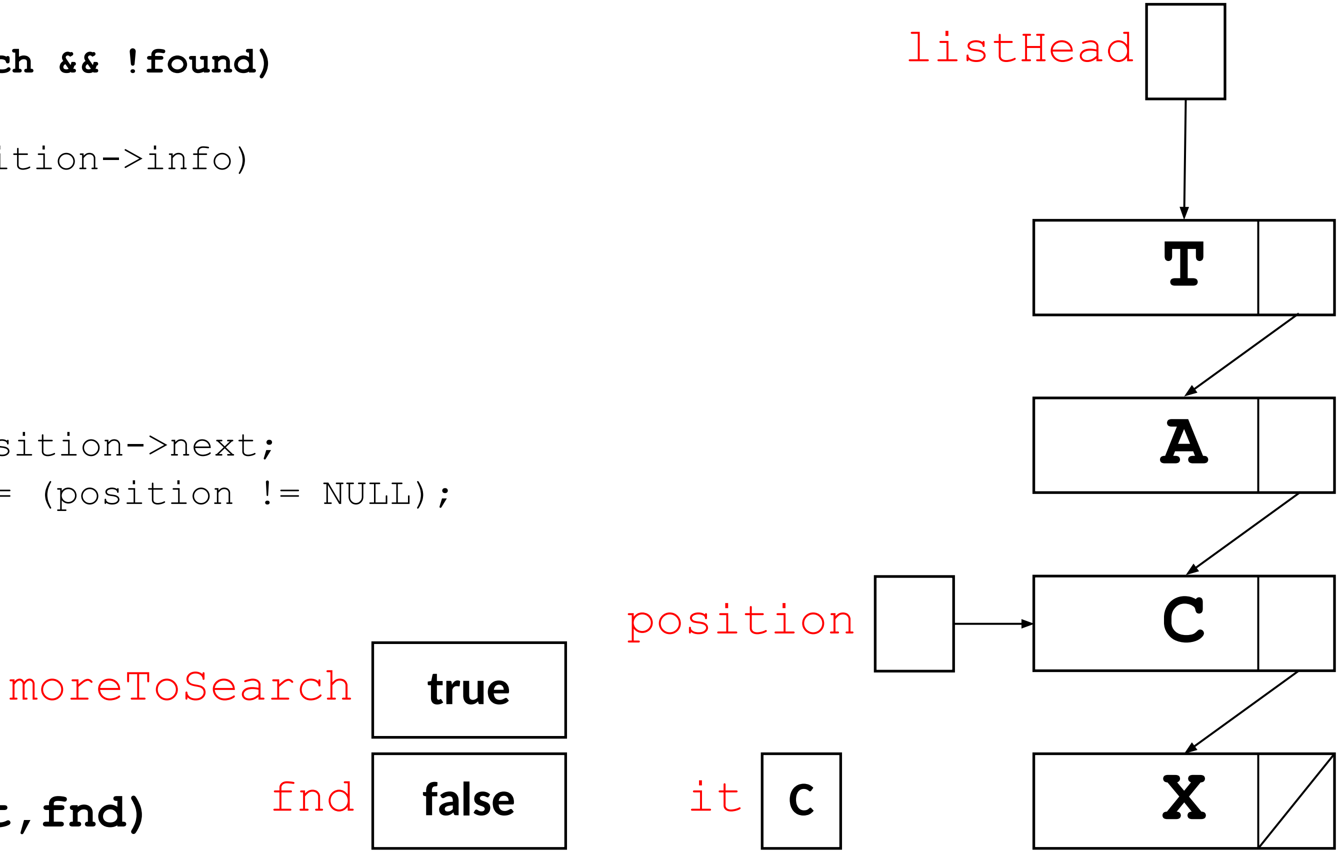
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* position = listHead;
  bool moreToSearch = (position != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == position->info)
    {
      found = true;
    }
    else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



RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

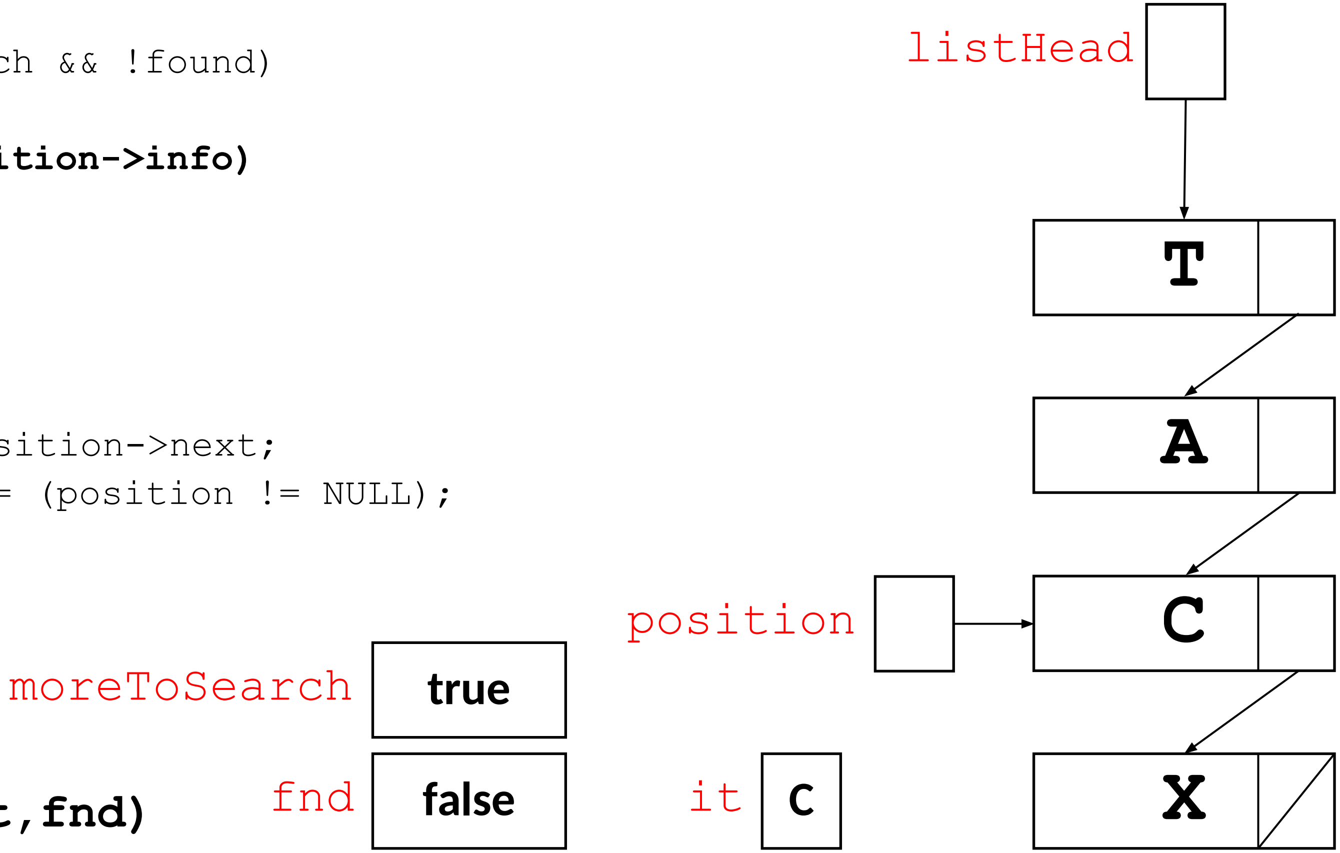
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* position = listHead;
  bool moreToSearch = (position != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == position->info)
    {
      found = true;
    }
    else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



RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

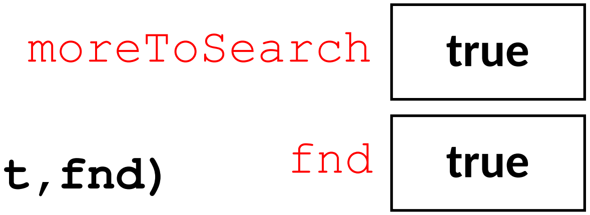
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* position = listHead;
  bool moreToSearch = (position != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == position->info)
    {
      found = true;
    }
    else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

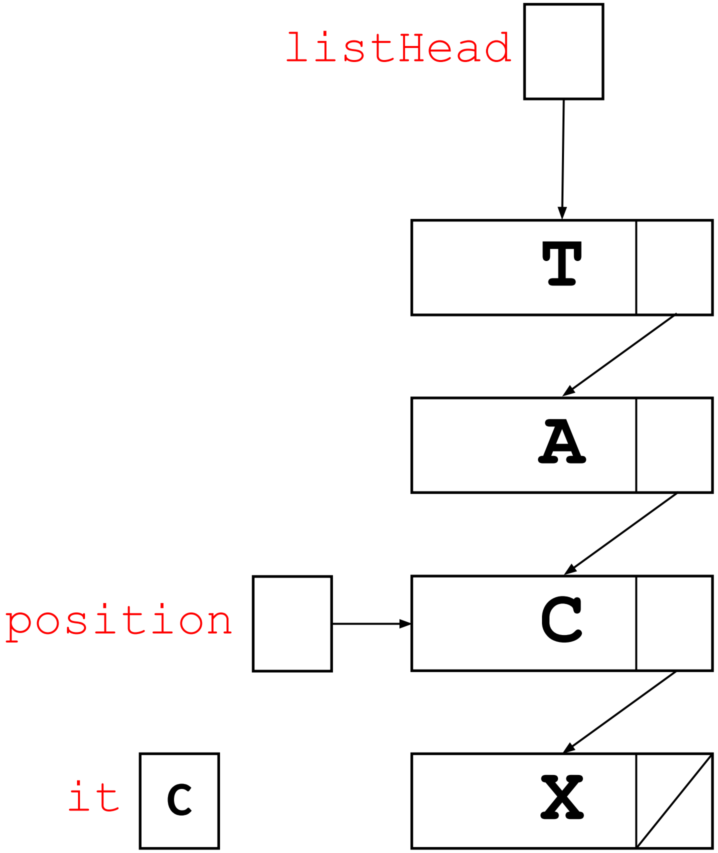


RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

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

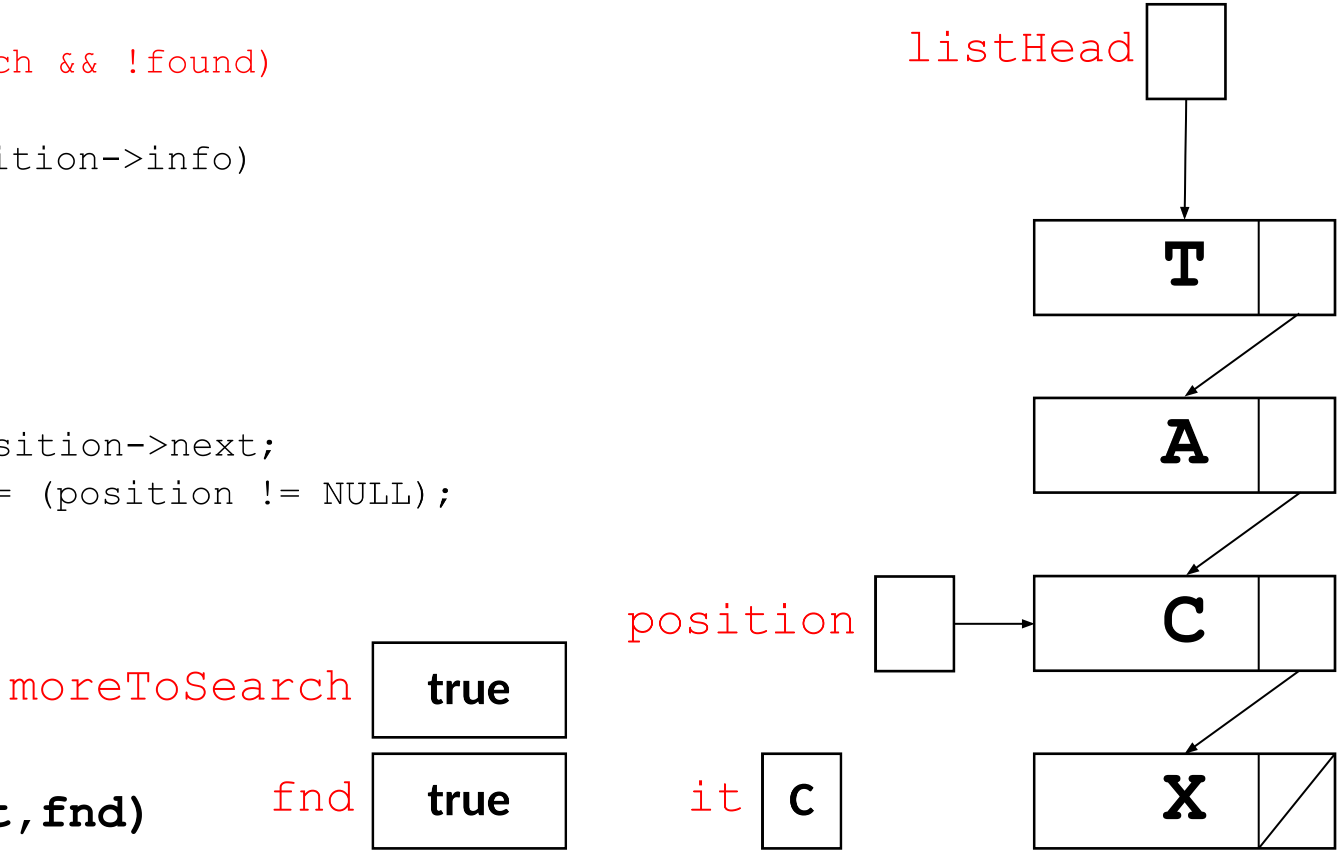




RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

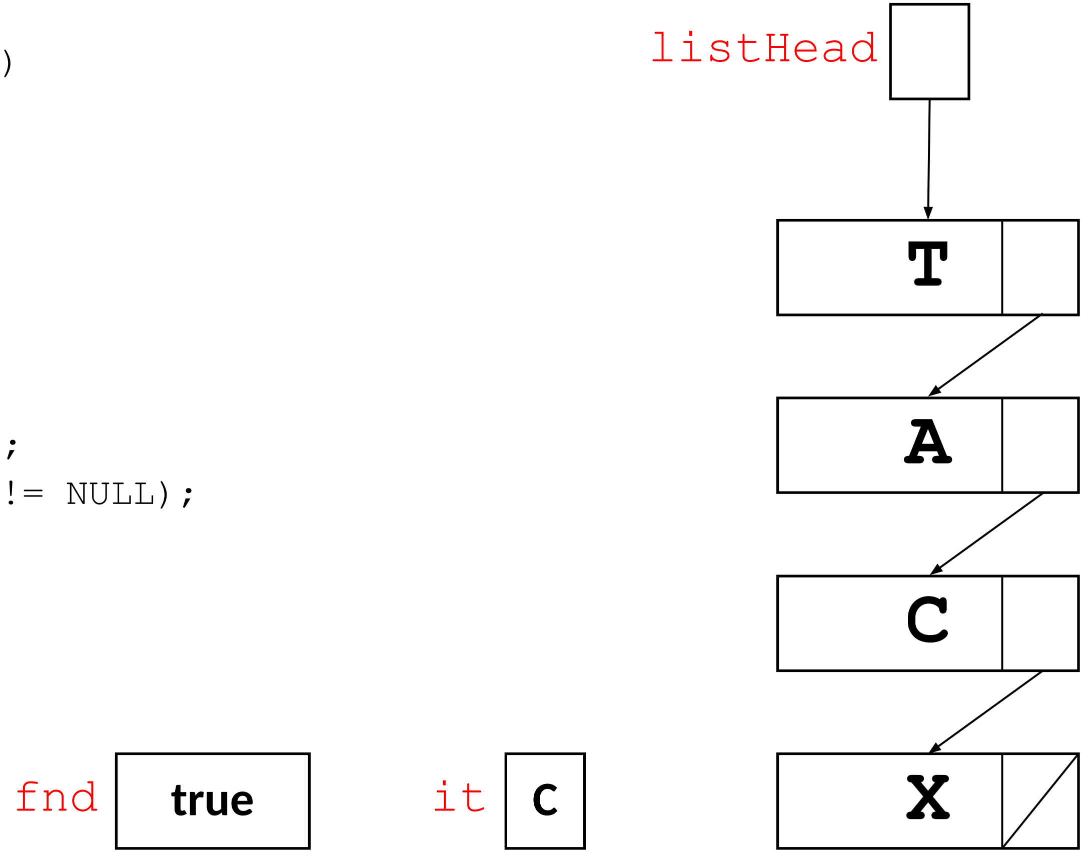
```
template <class ItemType>
void UnsortedType<ItemType>::RetrieveItem(ItemType& item, bool&
found)
{
  NodeType* position = listHead;
  bool moreToSearch = (position != NULL);
  found = false;
  while (moreToSearch && !found)
  {
    if (item == position->info)
    {
      found = true;
    }
    else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



RetrieveItem(it,fnd)

unsortedlinkedlist.cpp

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



unsortedlinkedlist.cpp

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

O(N)

unsortedlinkedlist.cpp

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

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

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

unsortedlinkedlist.cpp

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

  while (listHead != NULL)
  {
    tempPtr = listHead;
    listHead = listHead->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 = listHead;
  else
    currentPos = currentPos->next;
  item = currentPos->info;
}
```

unsortedlinkedlist.cpp

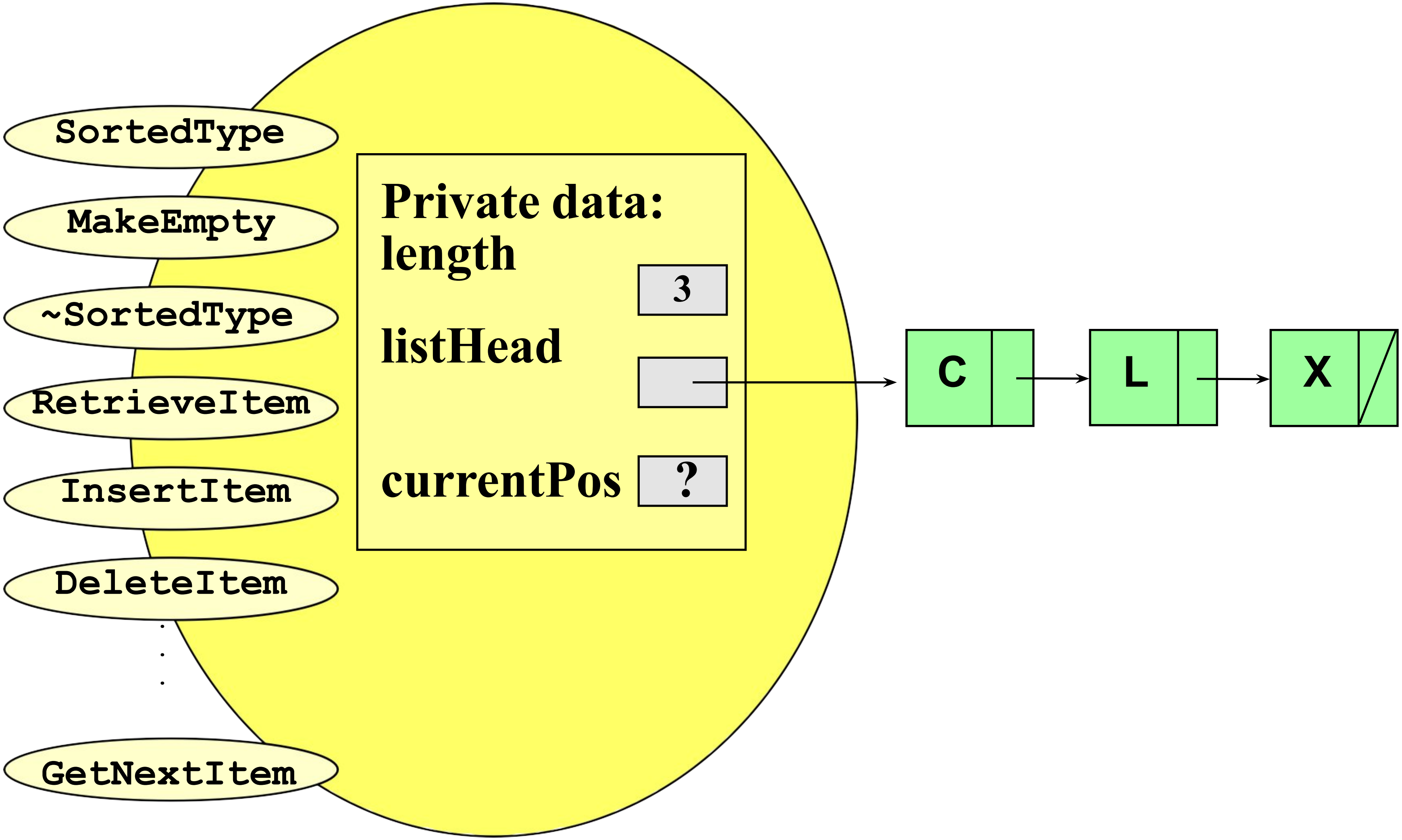
```
template <class ItemType>
void UnsortedType<ItemType>::ResetList()
{
  currentPos = NULL;
}
```

O(1)

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

O(1)

class SortedType<char>



SortedType

MakeEmpty

~SortedType

RetrieveItem

InsertItem

DeleteItem

.
.
.

GetNextItem

Private data:
length 3

listHead

currentPos ?

sortedlinkedlist.h

```
#ifndef SORTEDLINKEDLIST_H_INCLUDED
#define SORTEDLINKEDLIST_H_INCLUDED

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

public:
  SortedType();
  ~SortedType();
  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* listHead;
  int length;
  NodeType* currentPos;
};

#endif // SORTEDLINKEDLIST_H_INCLUDED
```

sortedlinkedlist.cpp

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

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

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

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

sortedlinkedlist.cpp

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

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

O(1)

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

O(1)

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

O(1)

sortedlinkedlist.cpp

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

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

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

sortedlinkedlist.cpp

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

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

O(N)

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

O(N)

sortedlinkedlist.cpp

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

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

sortedlinkedlist.cpp

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

O(1)

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

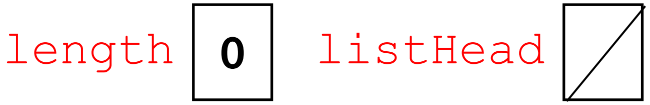
O(1)

sortedlinkedlist.cpp

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
  while (moreToSearch)
  {
    if (position->info < item)
    {
      predLoc = position;
      position = position->next;
      moreToSearch = (position != NULL);
    }
    else moreToSearch = false;
  }
```

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```

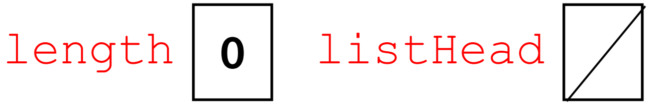
InsertItem(‘L’)



length 0 listHead

InsertItem(‘L’)

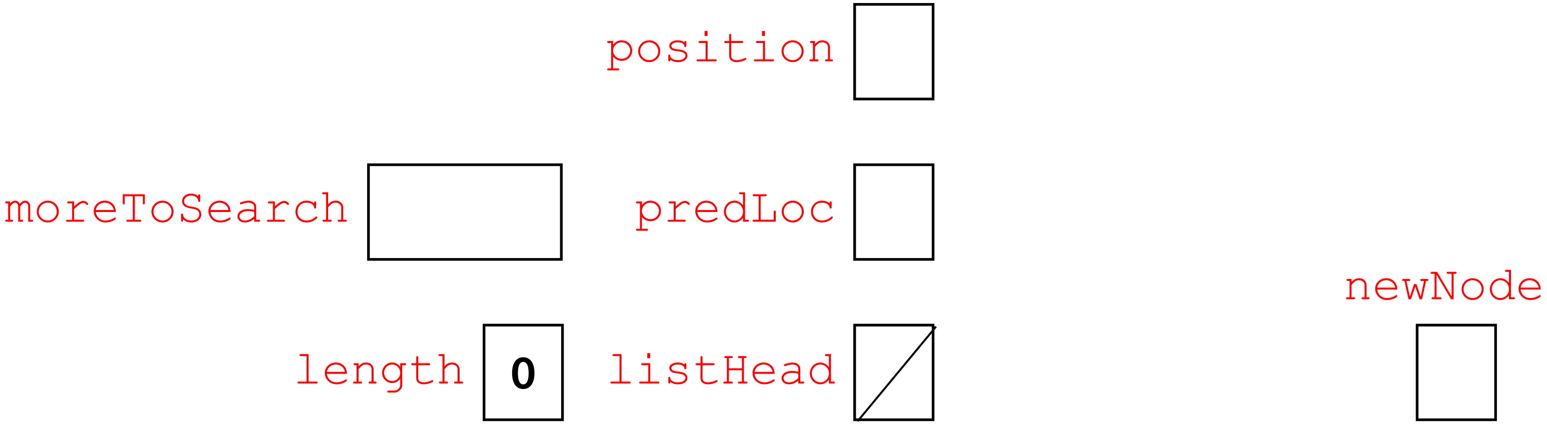
```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
```



length 0 listHead

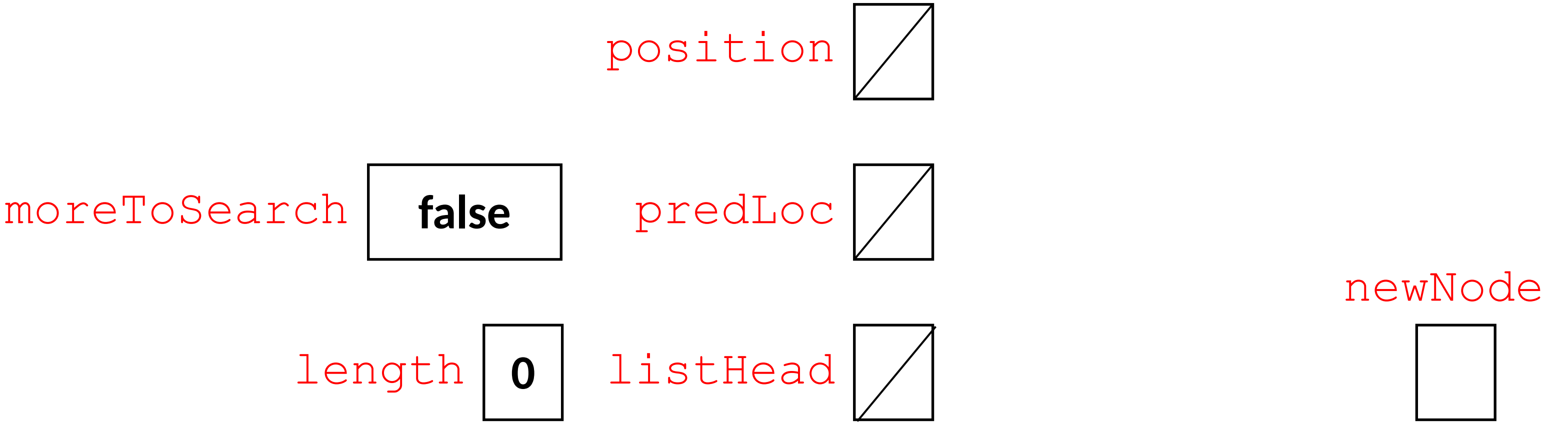
InsertItem('L')

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
```



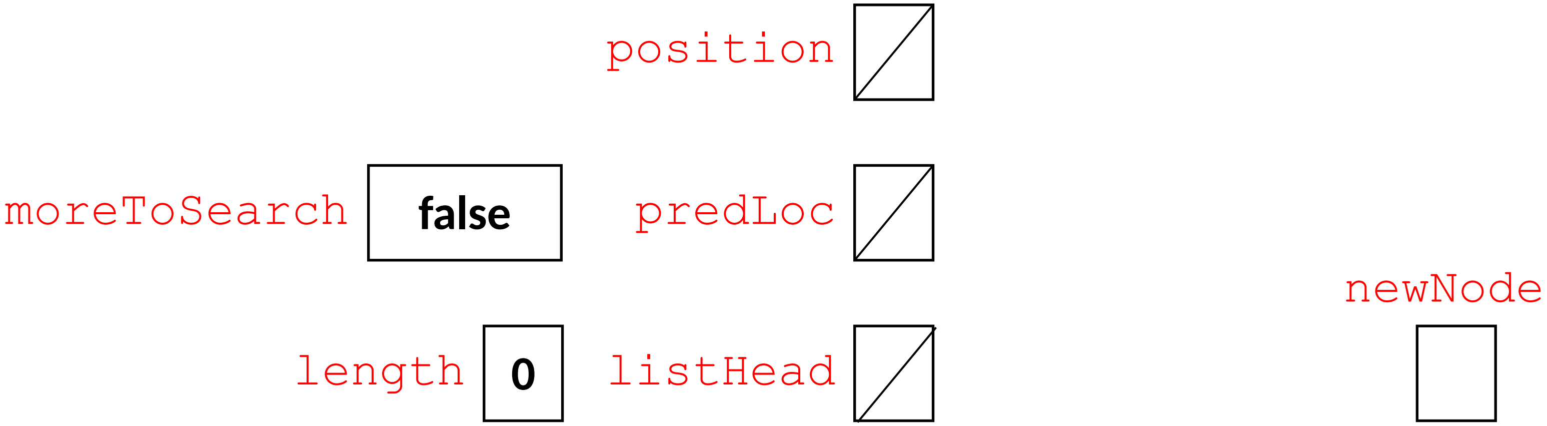
InsertItem('L')

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
```



InsertItem('L')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



position

moreToSearch false

predLoc

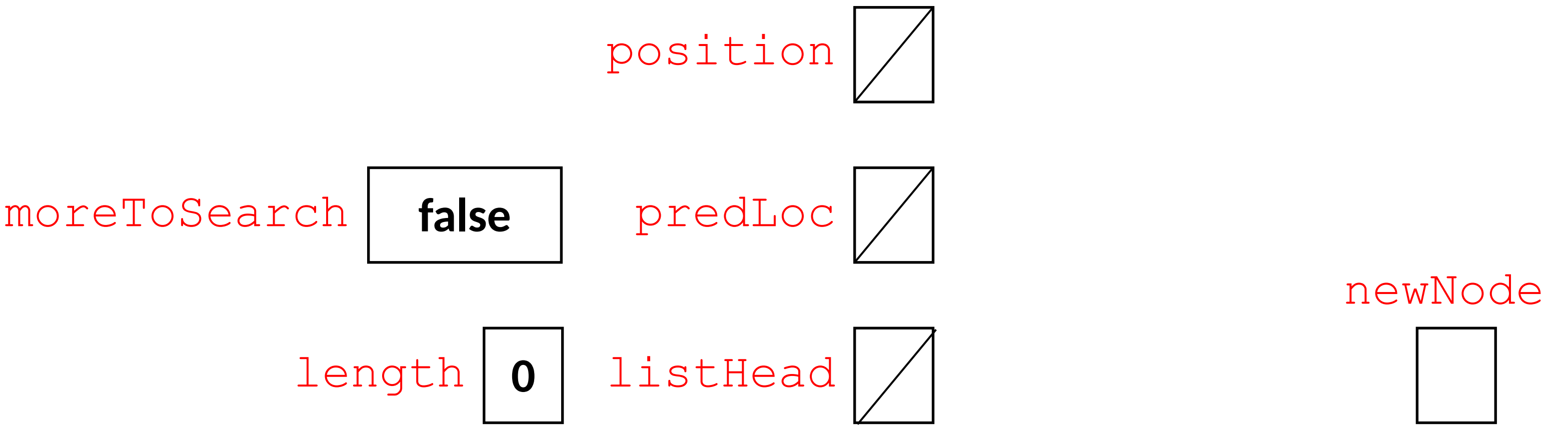
newNode

length 0

listHead

InsertItem('L')

```
while (moreToSearch)
  {
    if (position->info < item)
    {
      predLoc = position;
      position = position->next;
      moreToSearch = (position != NULL);
    }
    else moreToSearch = false;
  }
```



position

moreToSearch false

predLoc

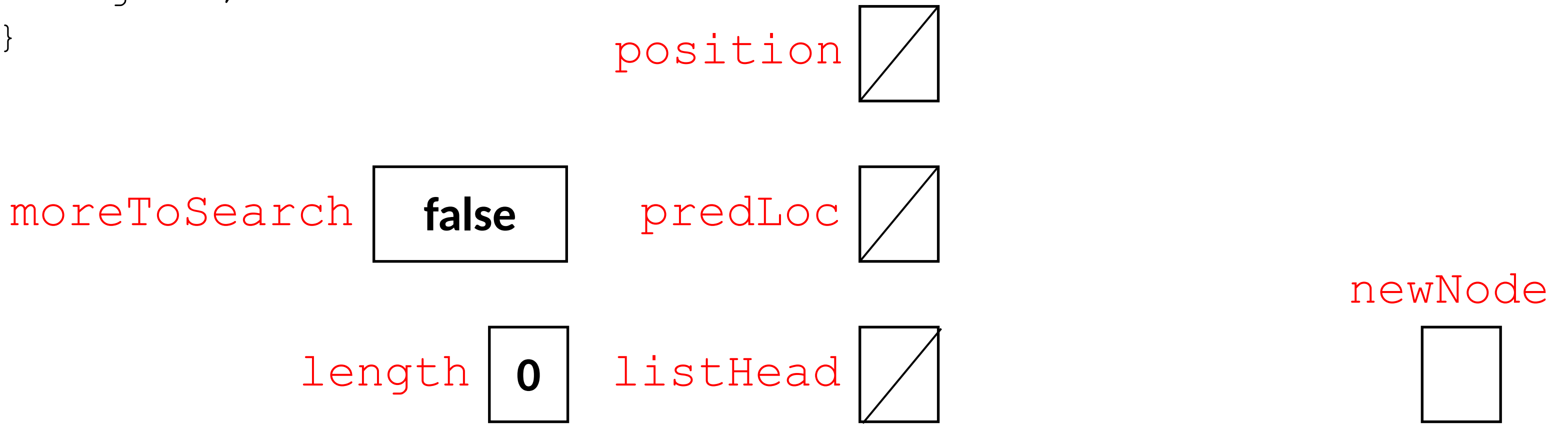
newNode

length 0

listHead

InsertItem('L')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



position

moreToSearch false

predLoc

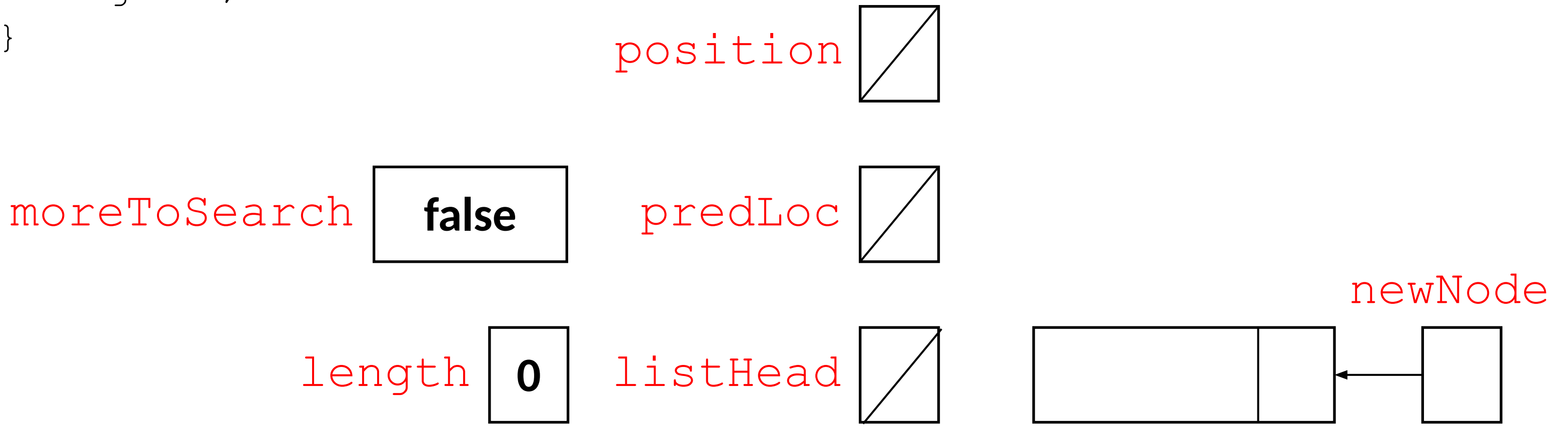
newNode

length 0

listHead

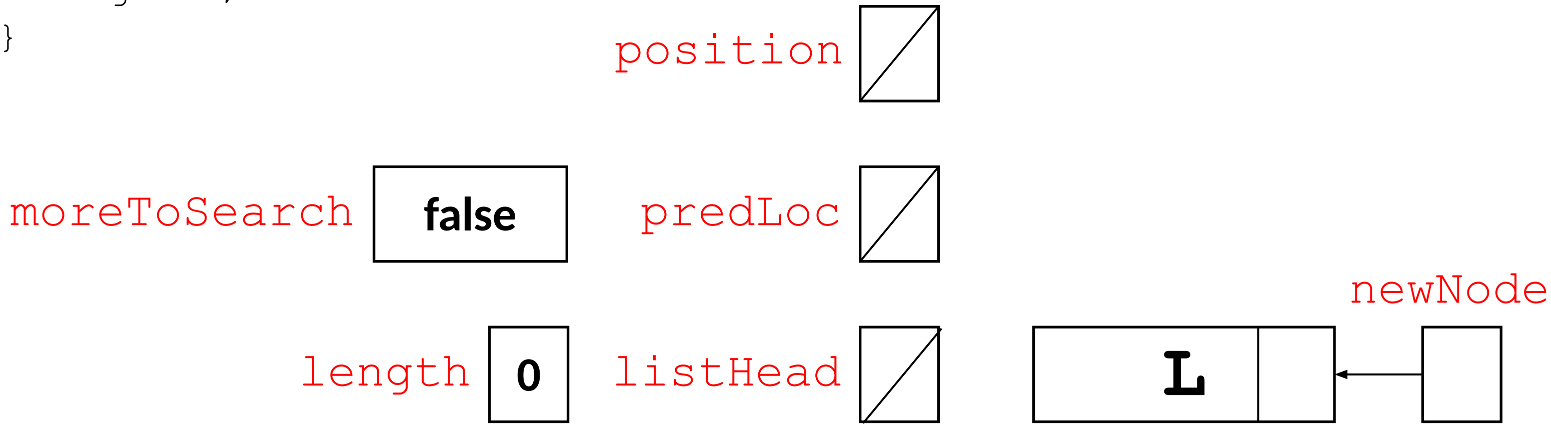
InsertItem('L')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



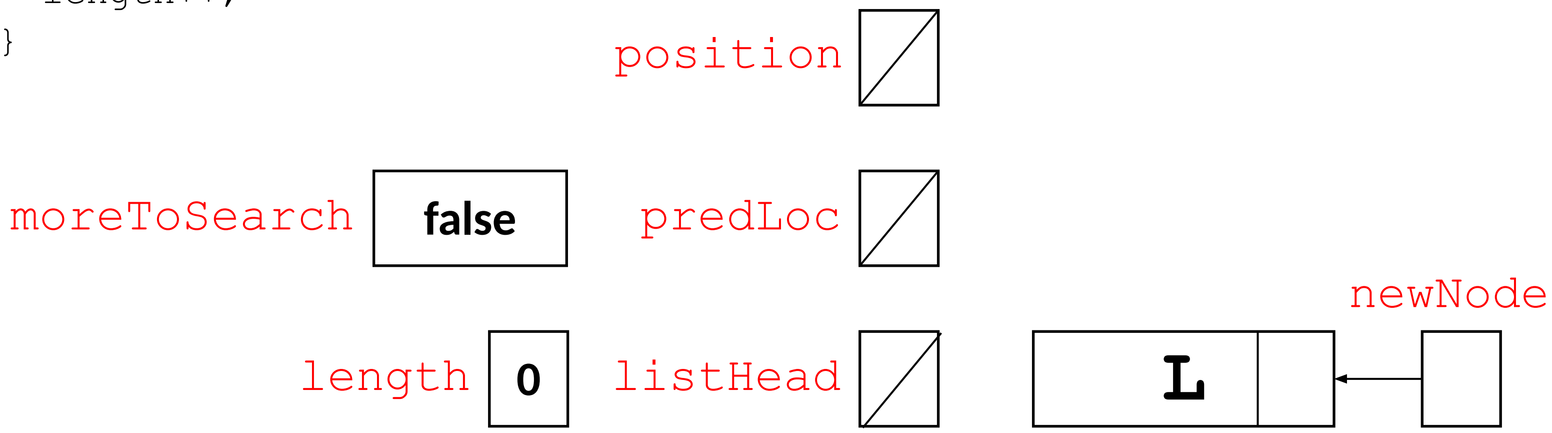
InsertItem(‘L’)

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



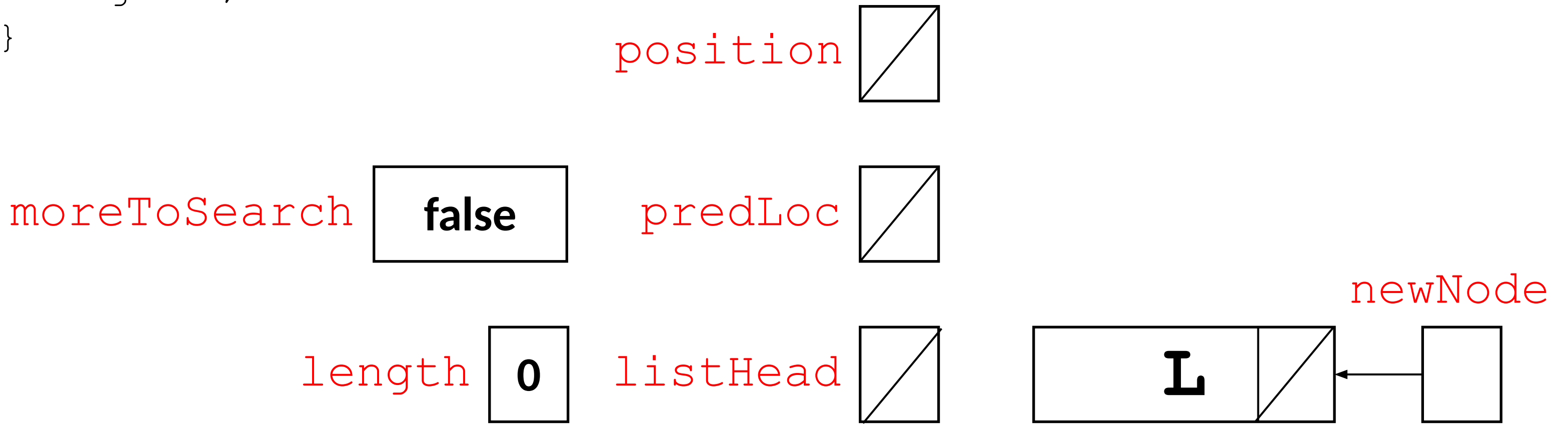
InsertItem('L')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



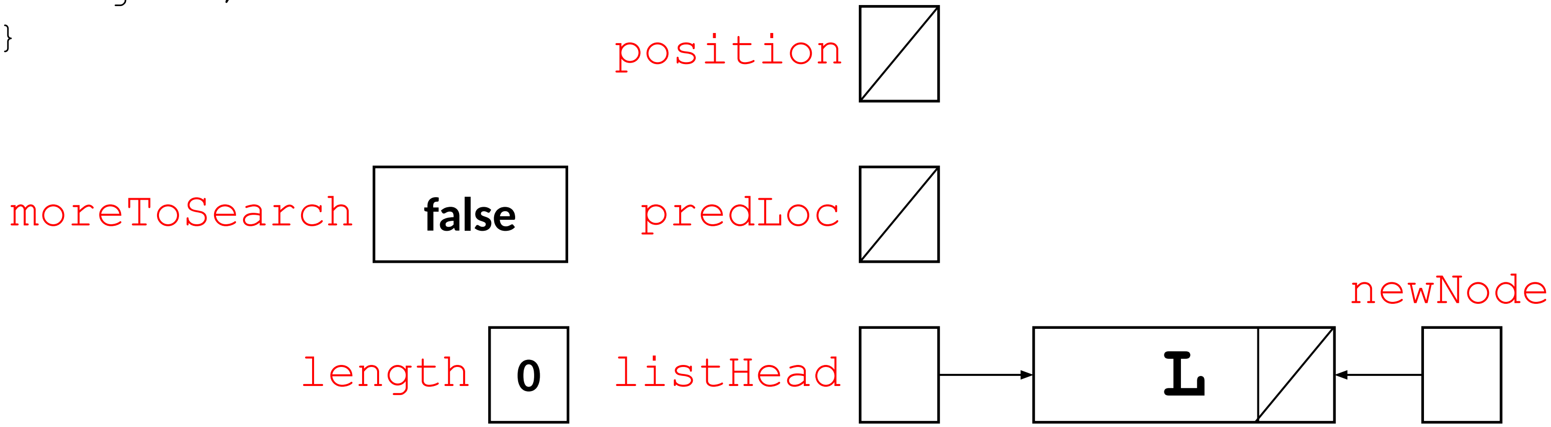
InsertItem('L')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



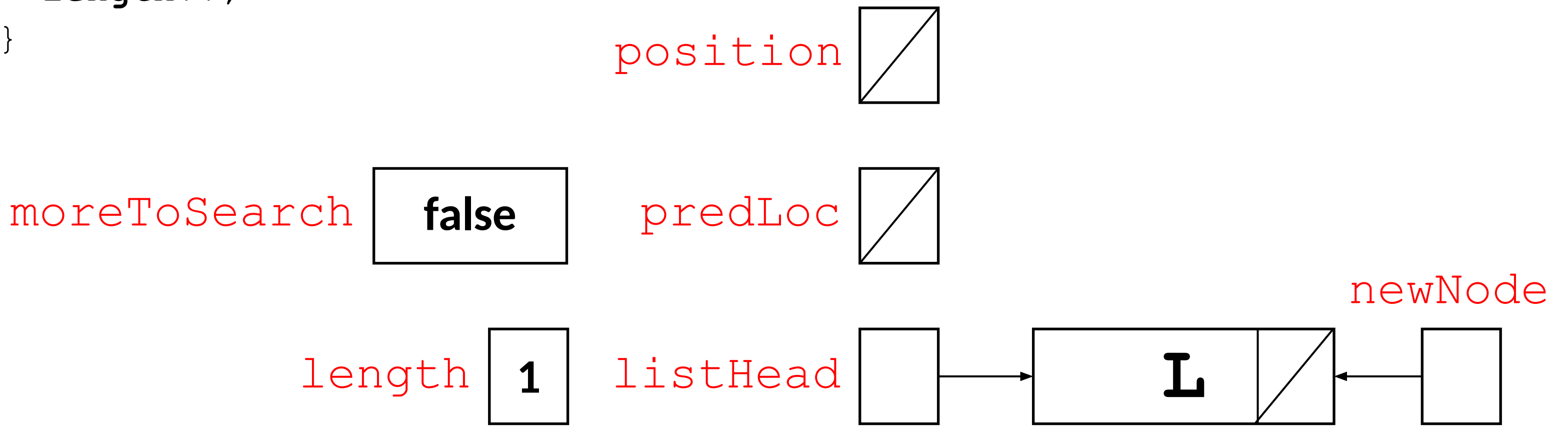
InsertItem('L')

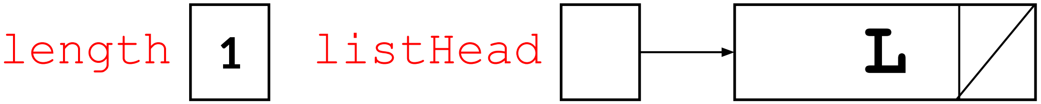
```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



InsertItem('L')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```





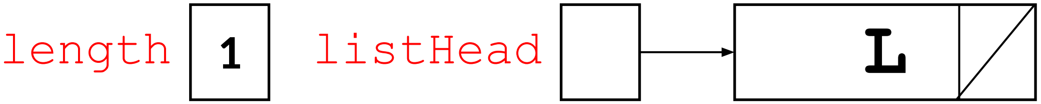
length

1

listHead

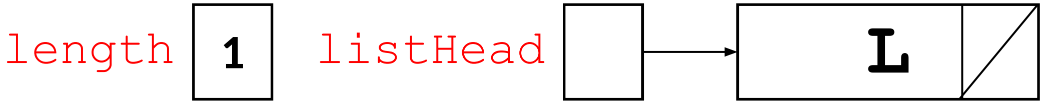
L

InsertItem('A')



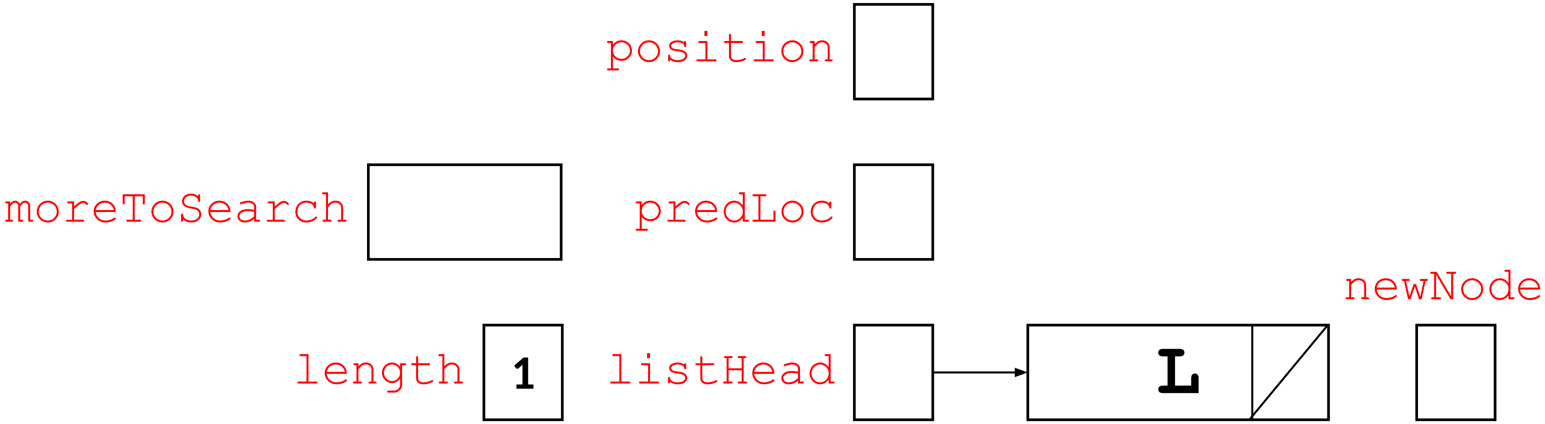
InsertItem(‘A’)

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
```



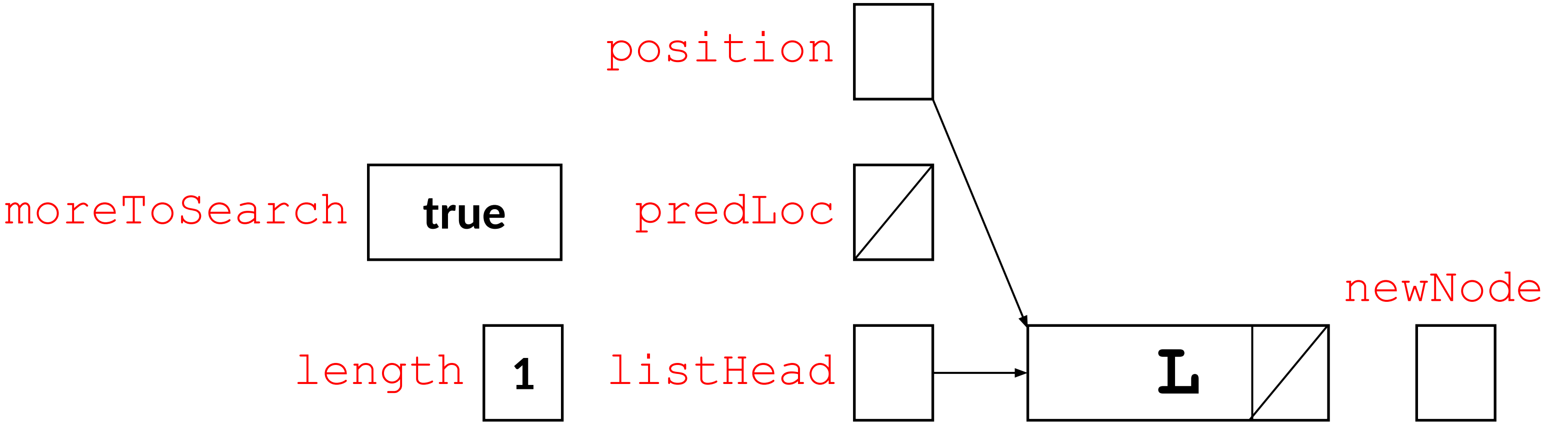
InsertItem('A')

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
```



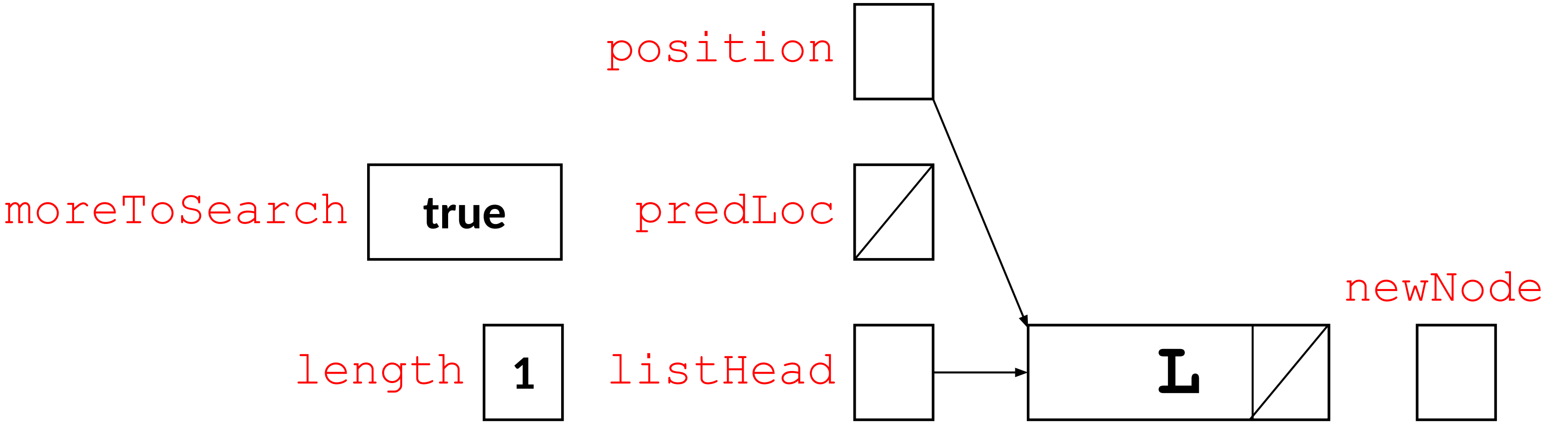
InsertItem('A')

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
```



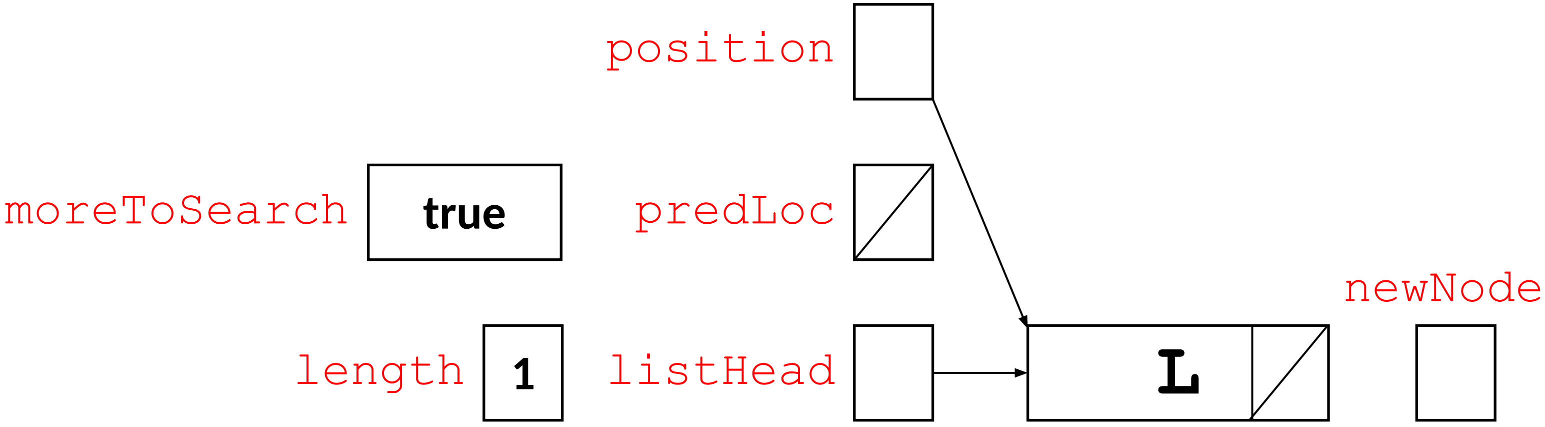
InsertItem('A')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



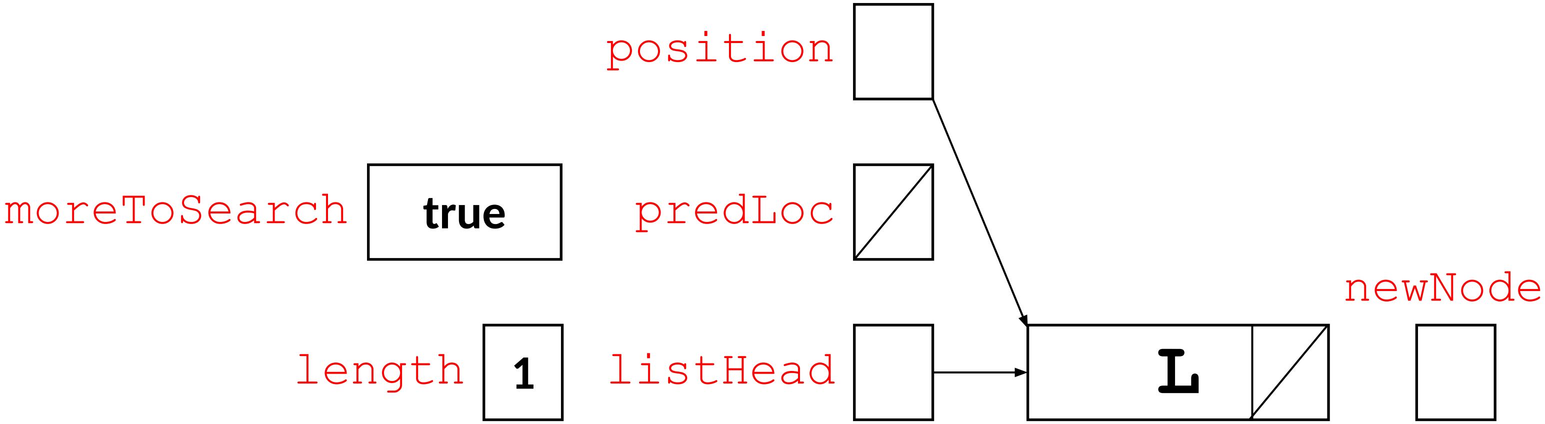
InsertItem('A')

```
while (moreToSearch)
  {
    if (position->info < item)
    {
      predLoc = position;
      position = position->next;
      moreToSearch = (position != NULL);
    }
    else moreToSearch = false;
  }
```



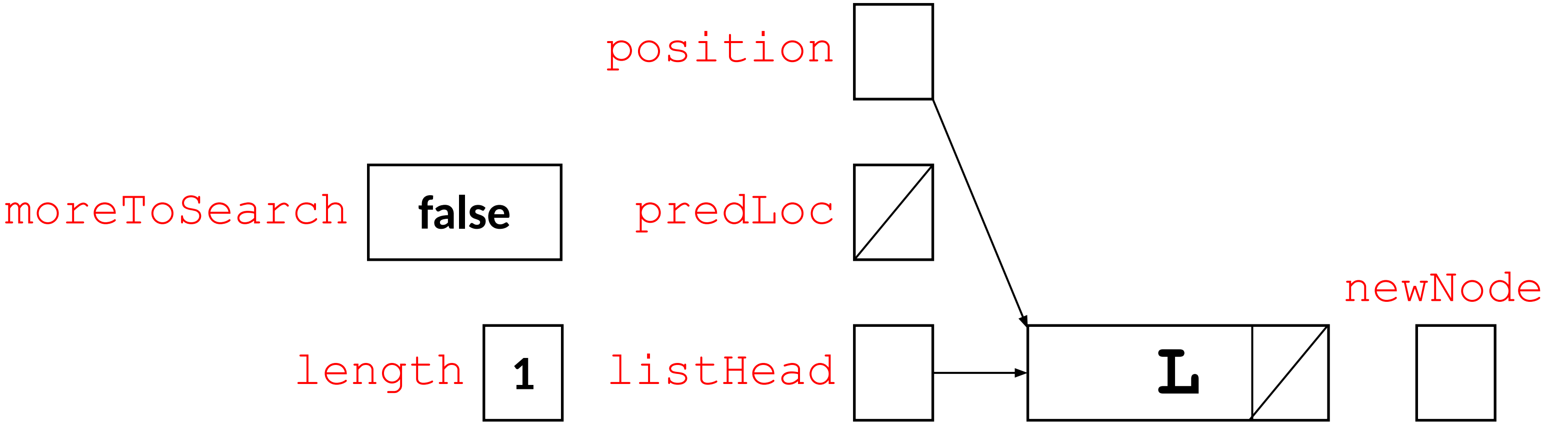
InsertItem('A')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



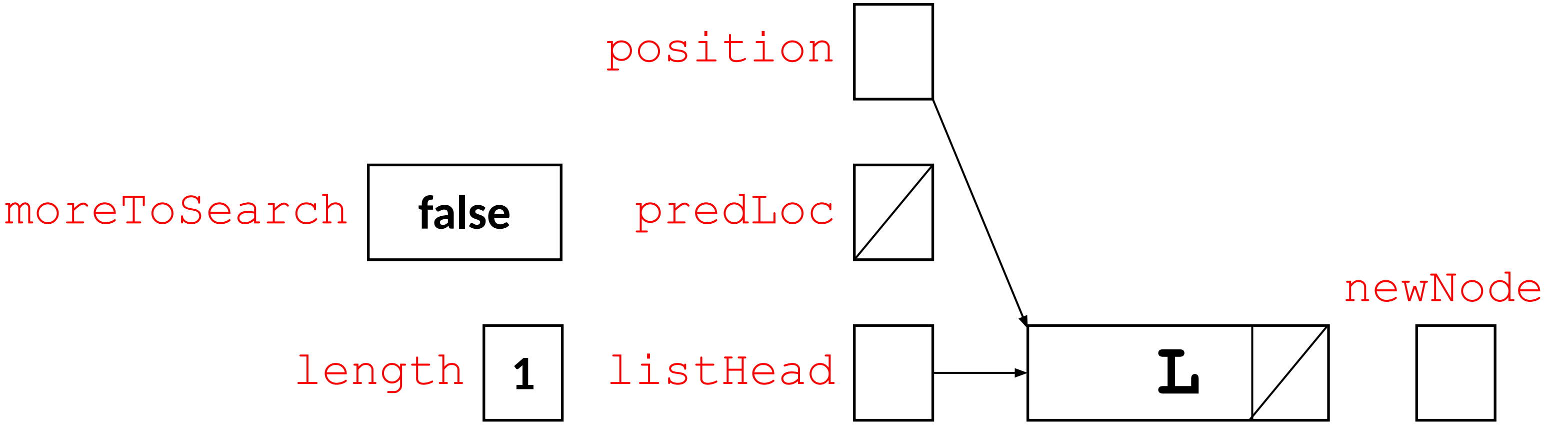
InsertItem('A')

```
while (moreToSearch)
  {
    if (position->info < item)
    {
      predLoc = position;
      position = position->next;
      moreToSearch = (position != NULL);
    }
    else moreToSearch = false;
  }
```



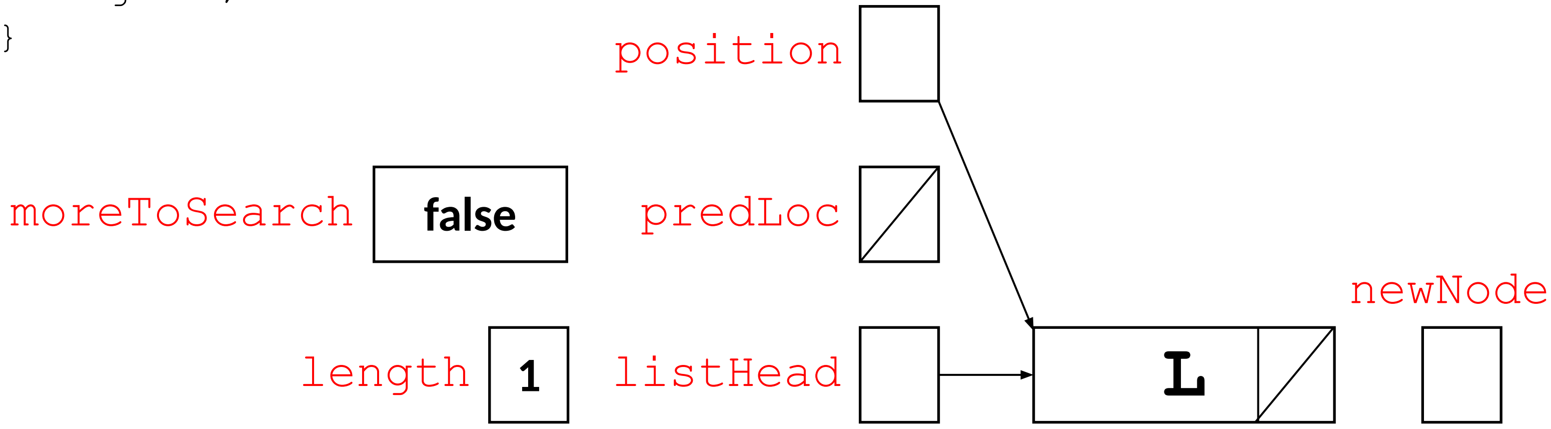
InsertItem('A')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



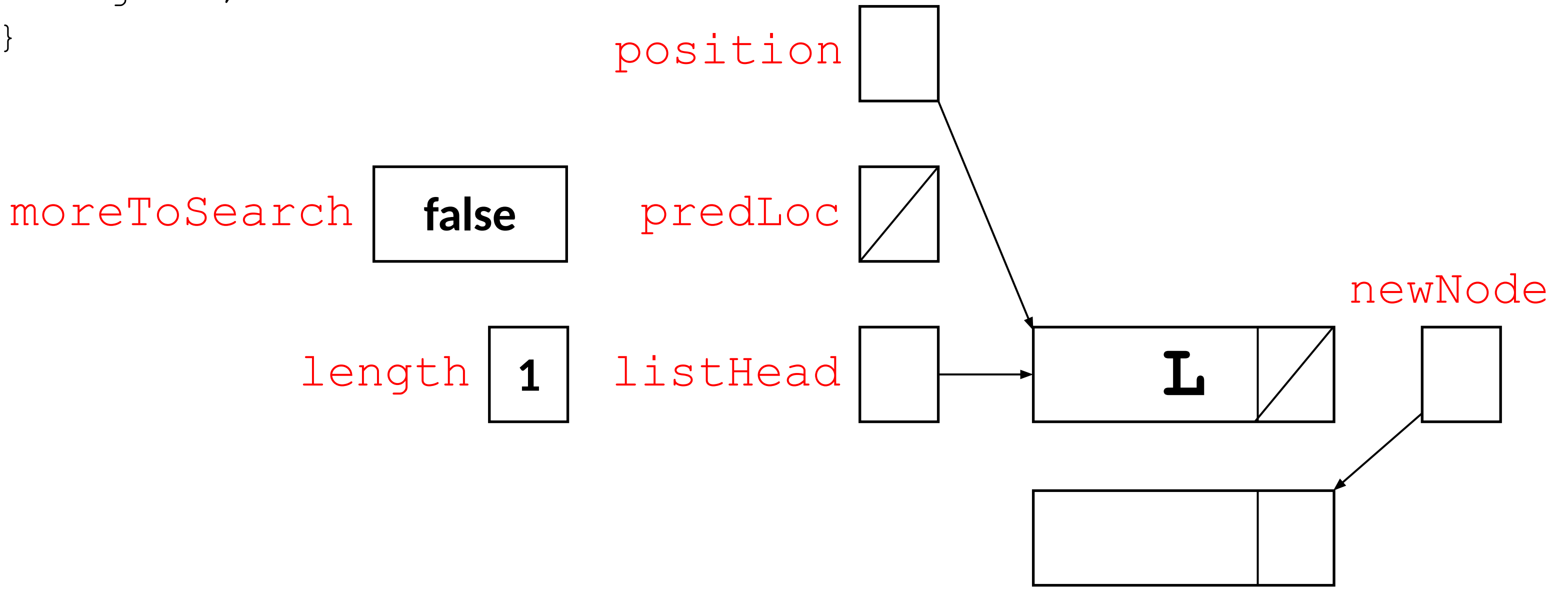
InsertItem('A')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



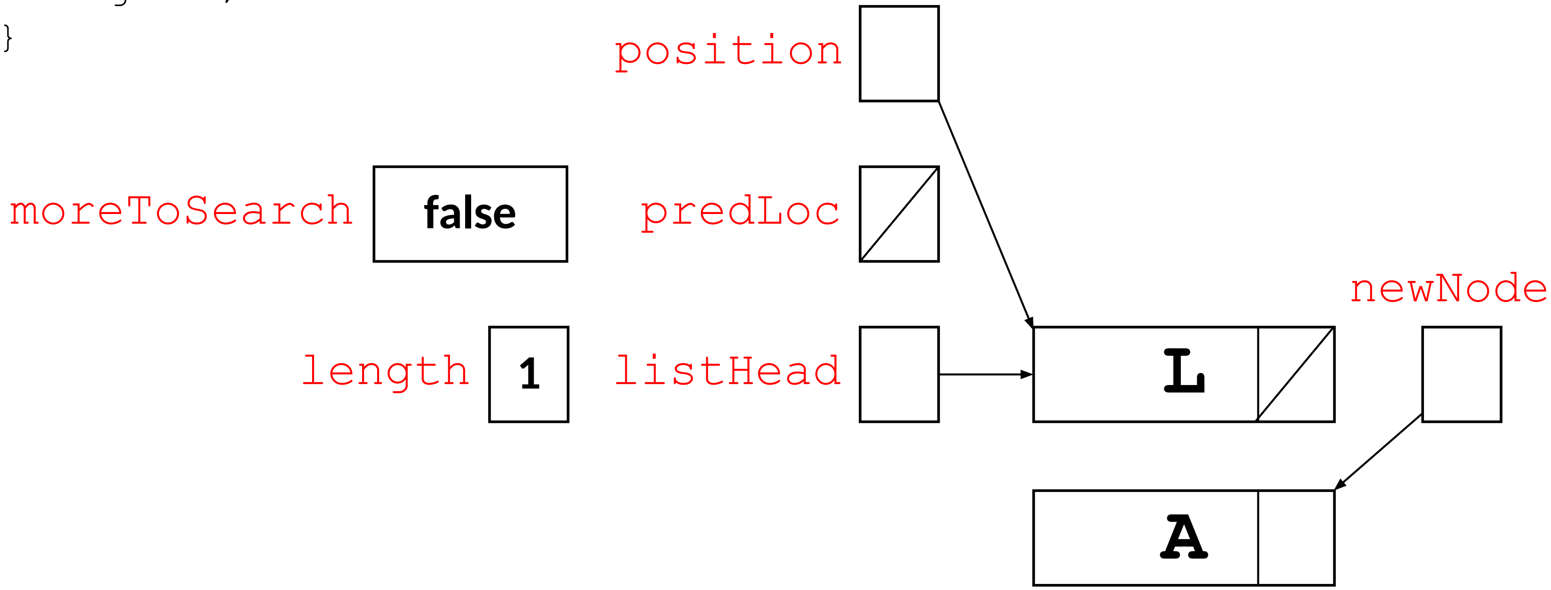
InsertItem('A')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



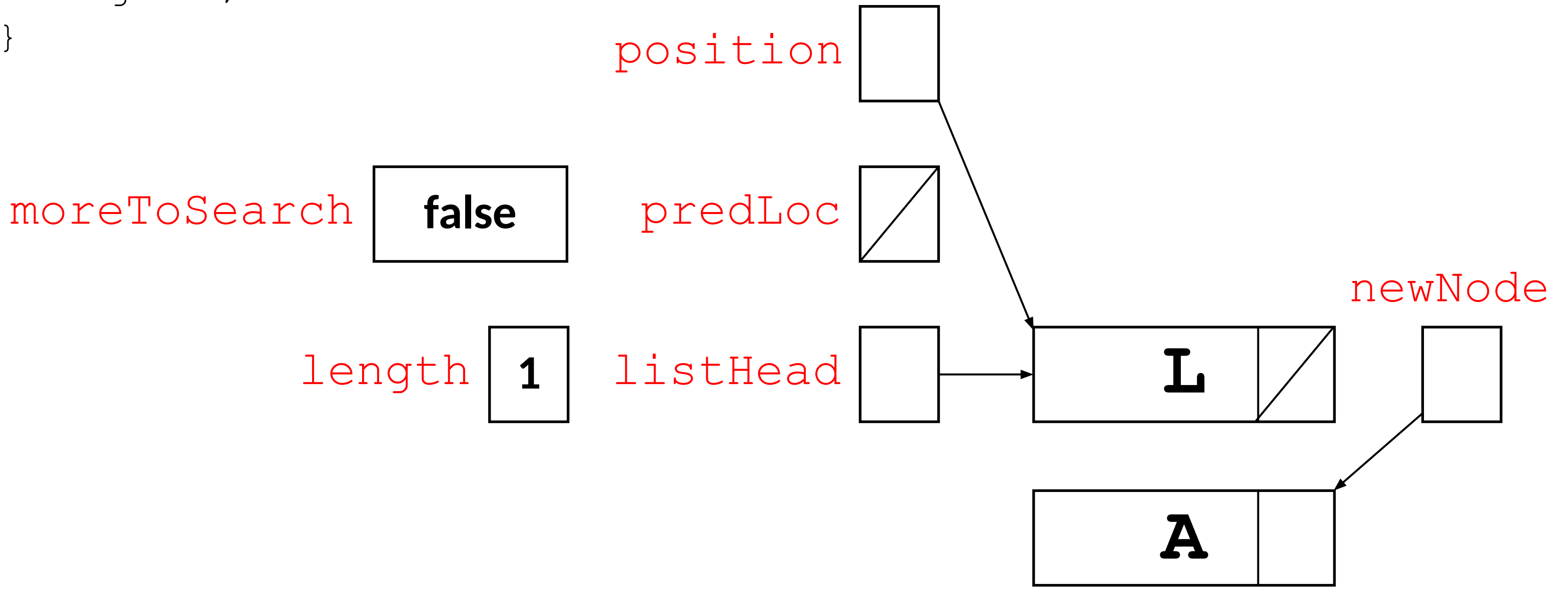
InsertItem('A')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



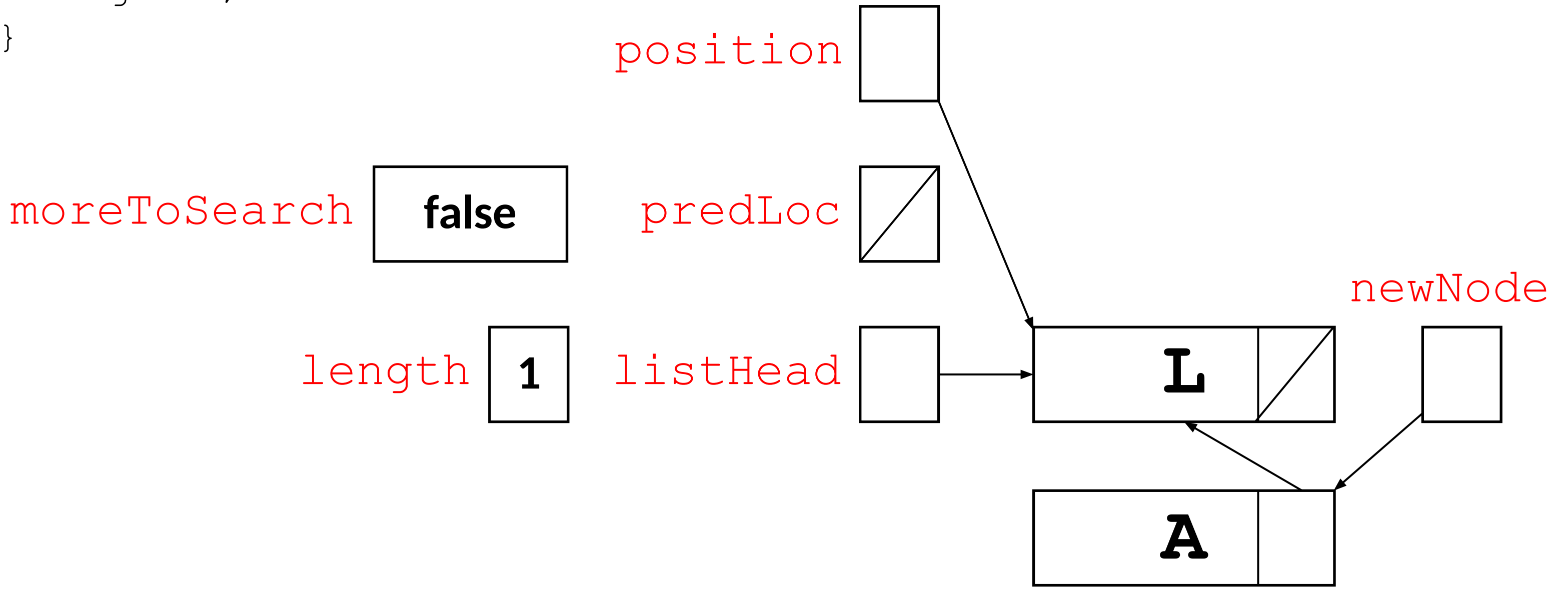
InsertItem('A')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



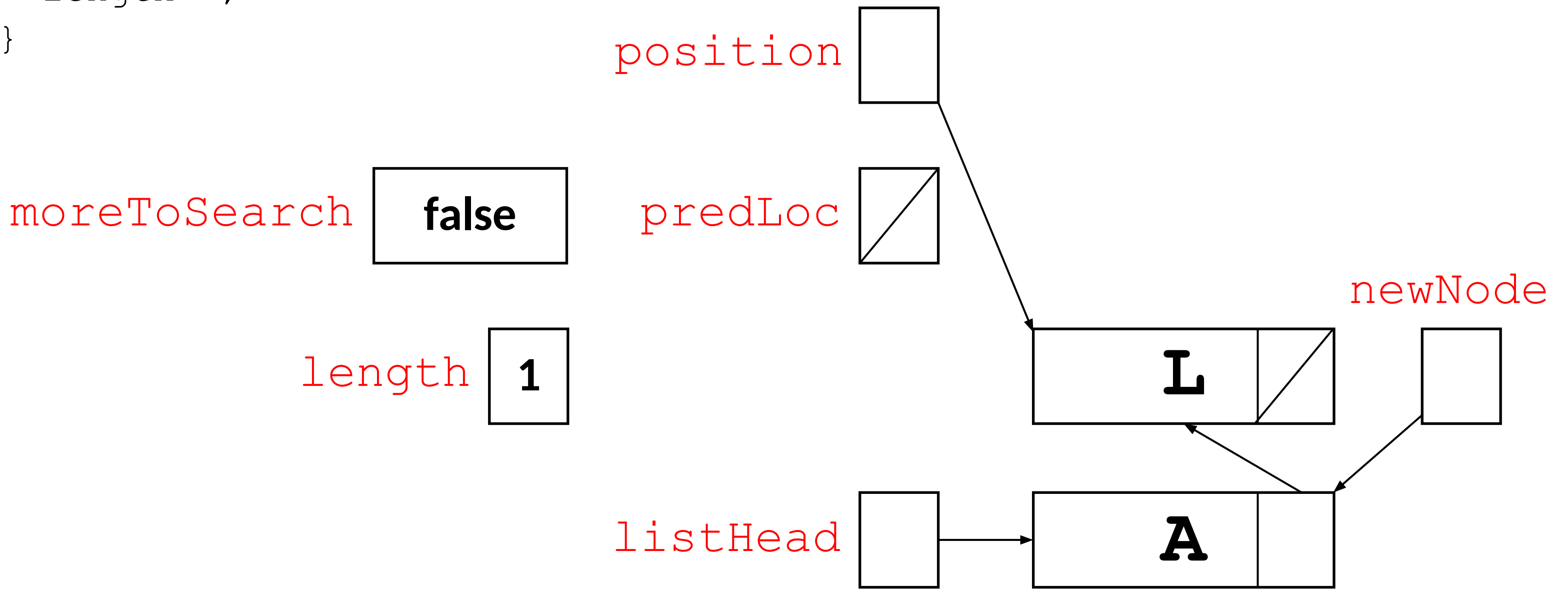
InsertItem('A')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



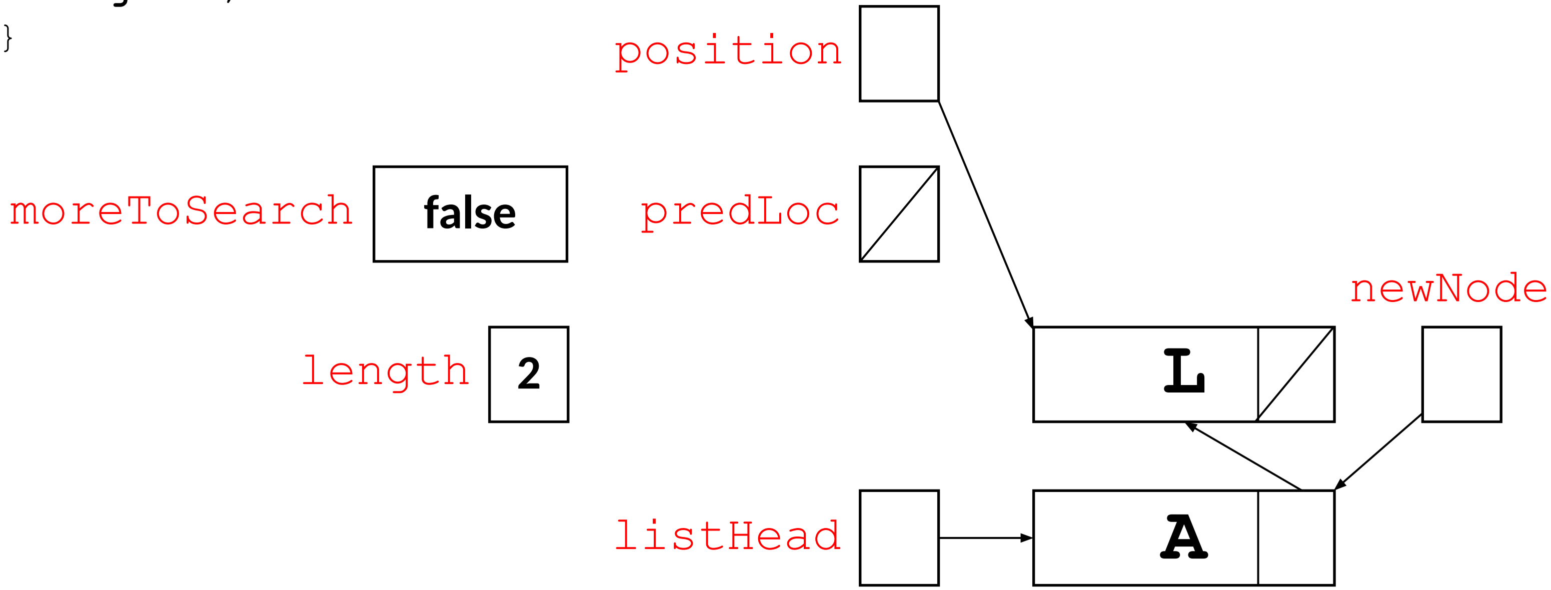
InsertItem('A')

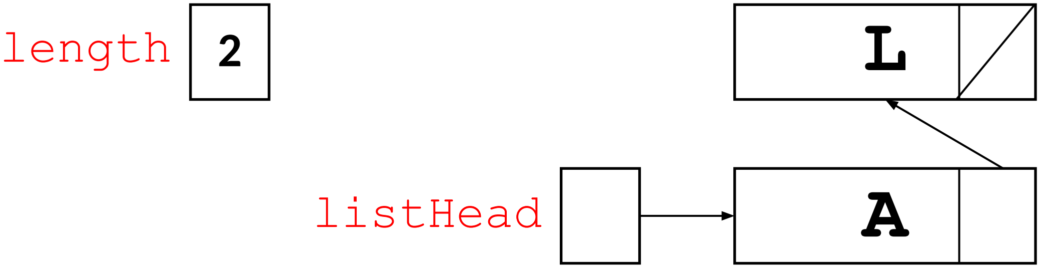
```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



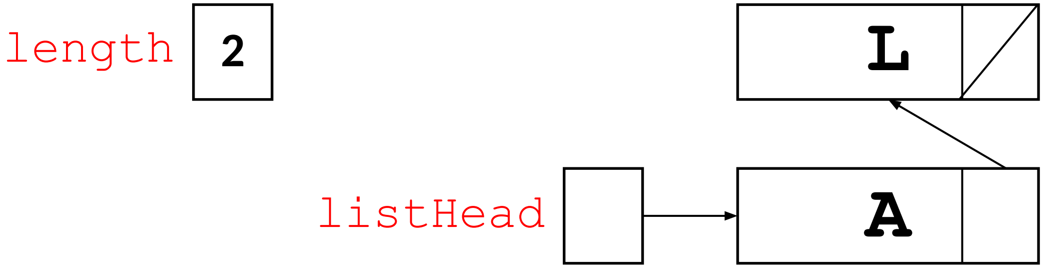
InsertItem('A')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



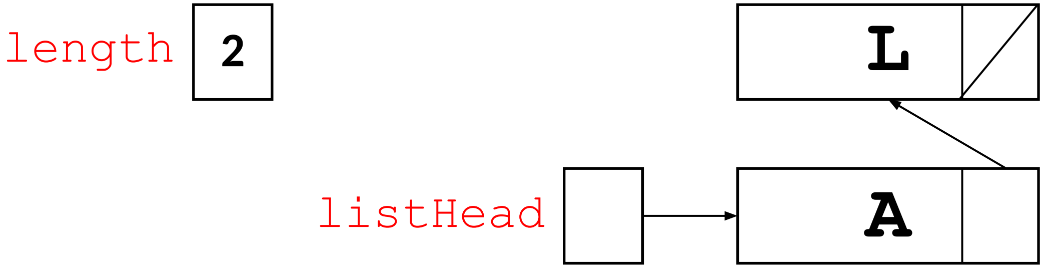

length
2
L
listHead
A

InsertItem('S')



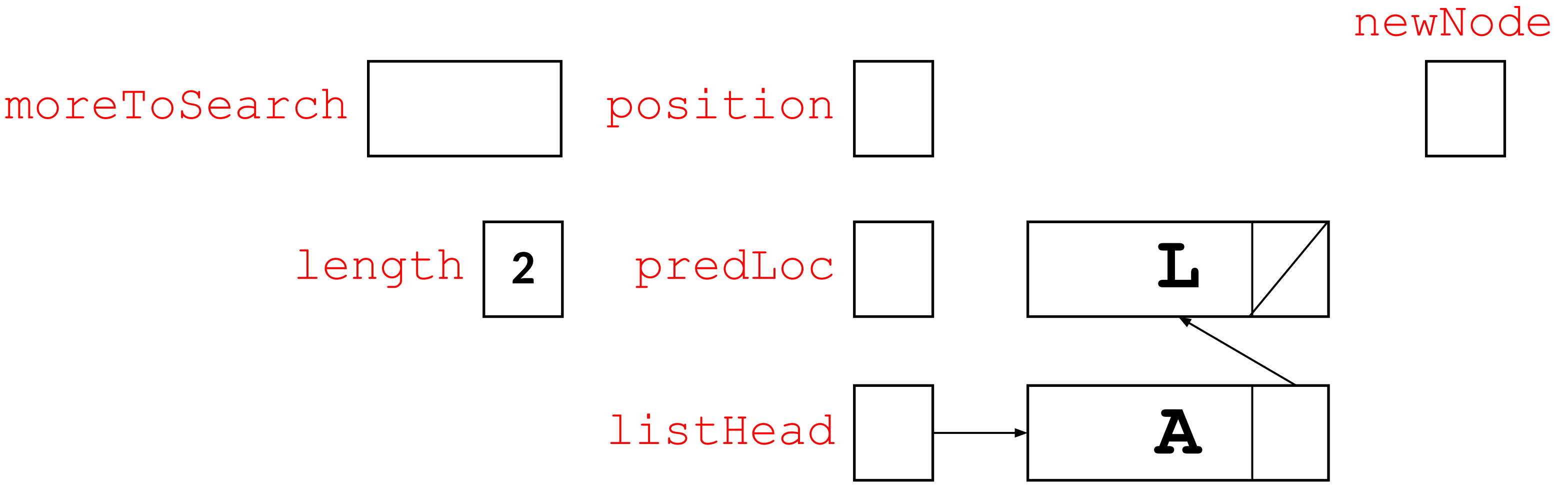
InsertItem('S')

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
```



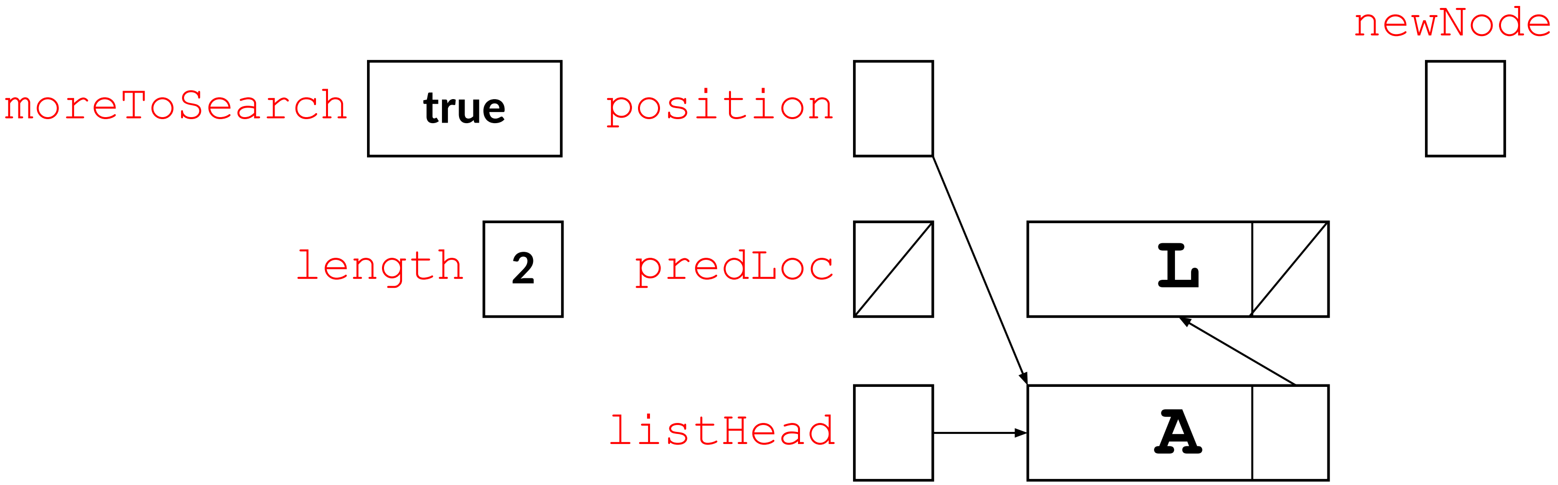
InsertItem('S')

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
```



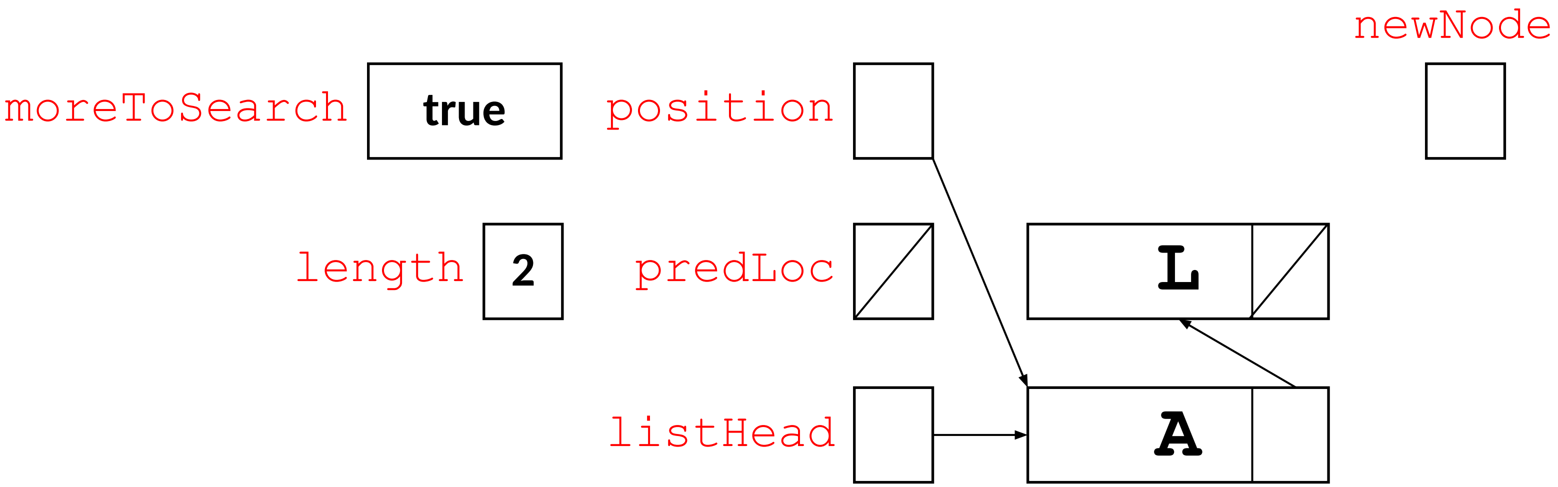
InsertItem('S')

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
```



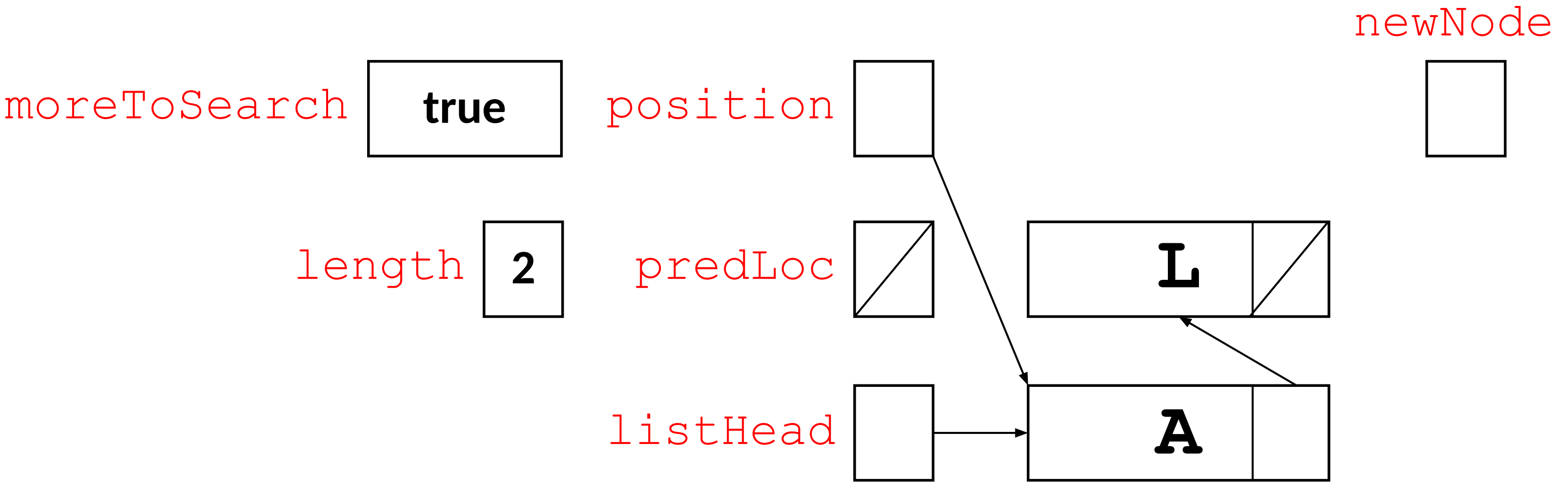
InsertItem('S')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



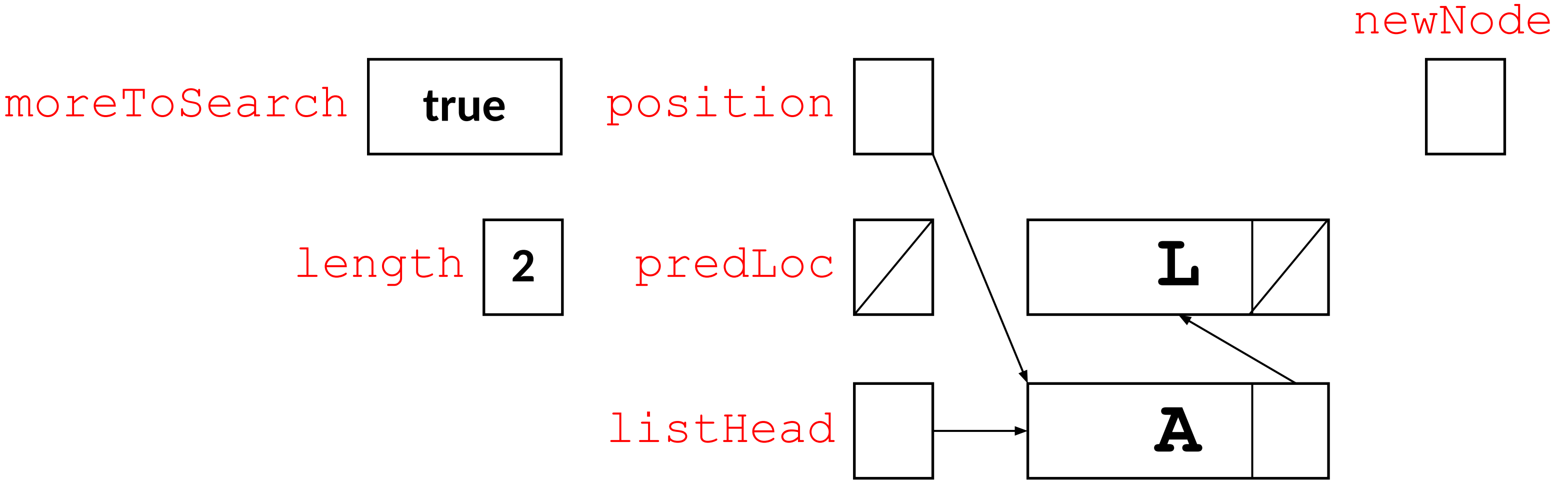
InsertItem('S')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



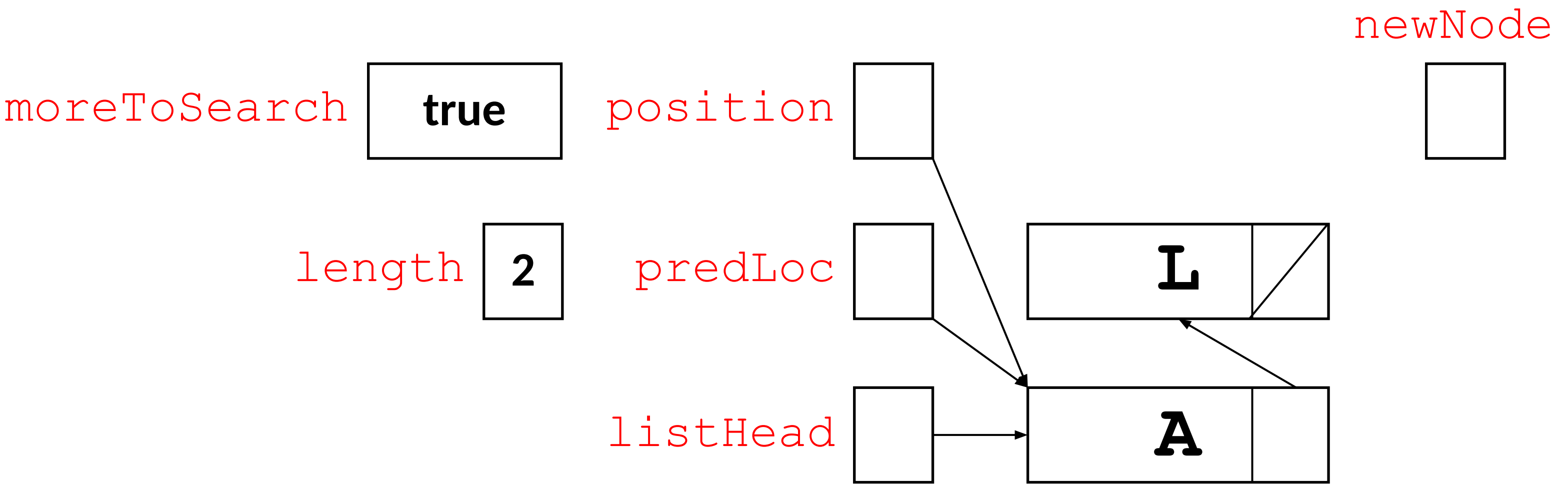
InsertItem('S')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



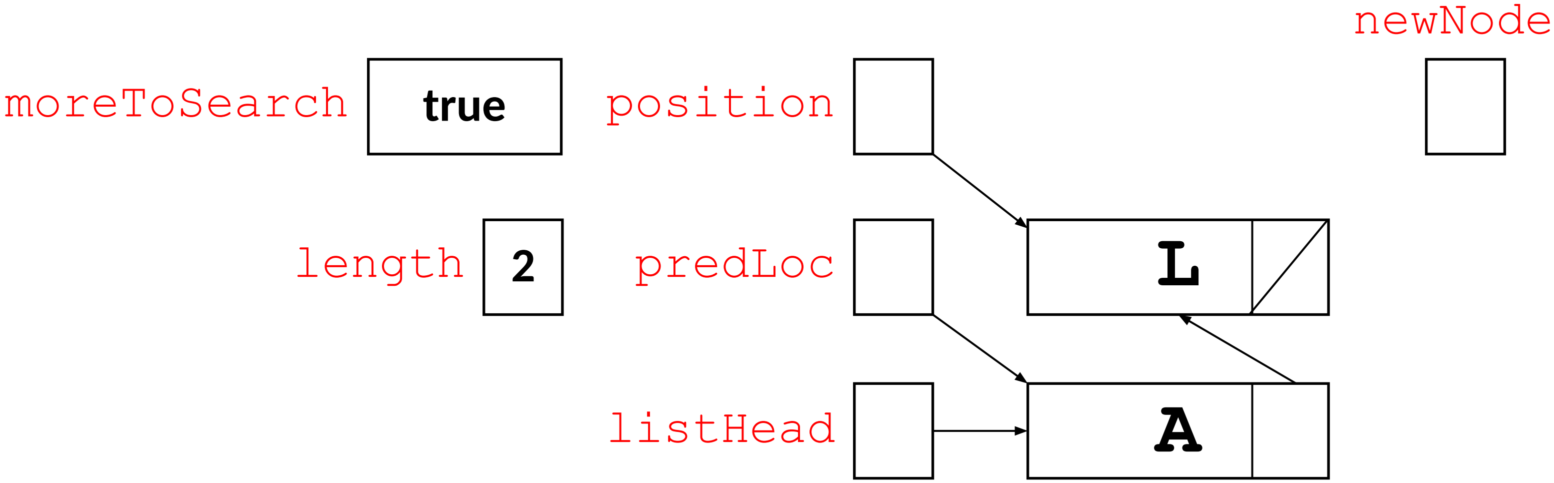
InsertItem('S')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



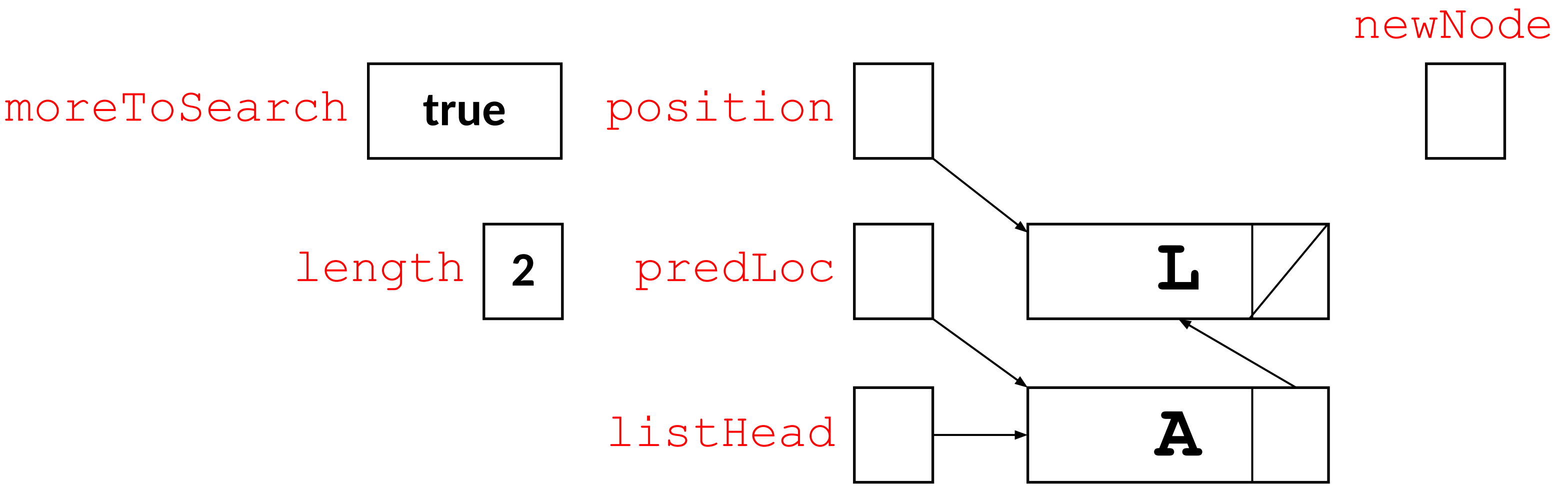
InsertItem('S')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



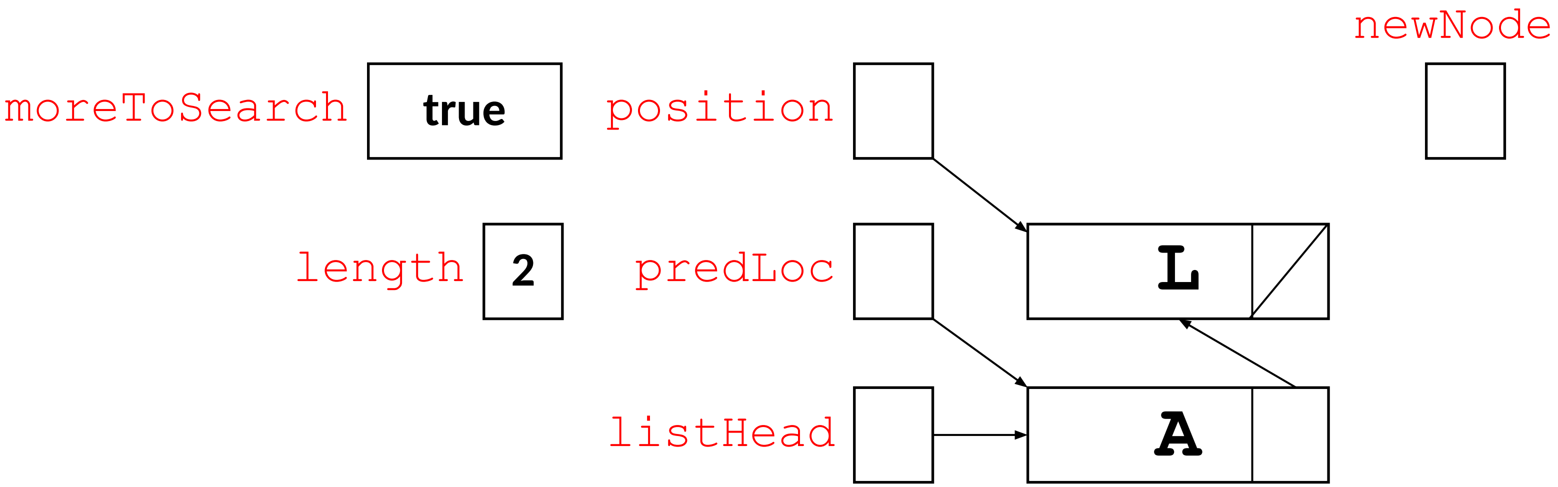
InsertItem('S')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



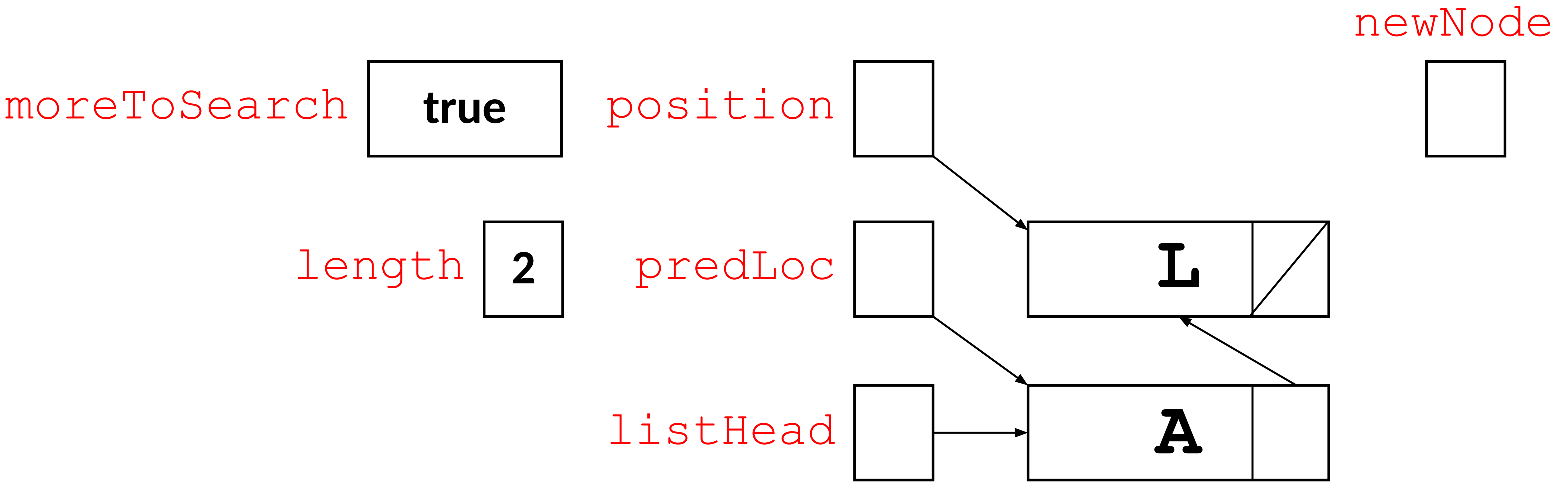
InsertItem('S')

```
while (moreToSearch)
  {
    if (position->info < item)
    {
      predLoc = position;
      position = position->next;
      moreToSearch = (position != NULL);
    }
    else moreToSearch = false;
  }
```



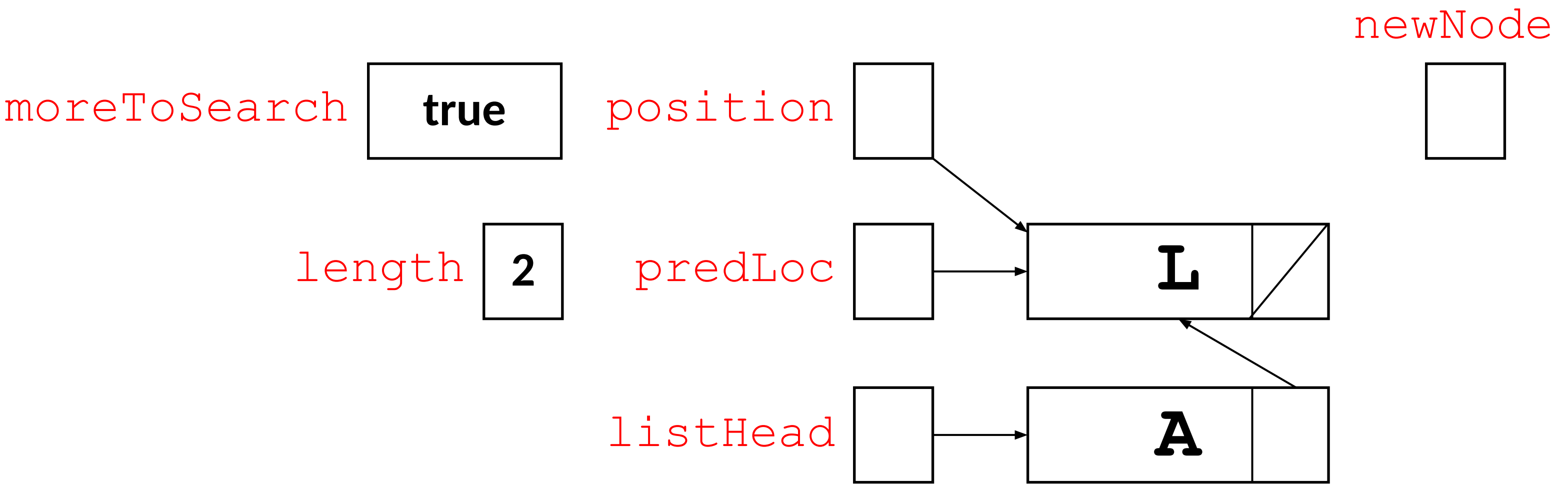
InsertItem('S')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



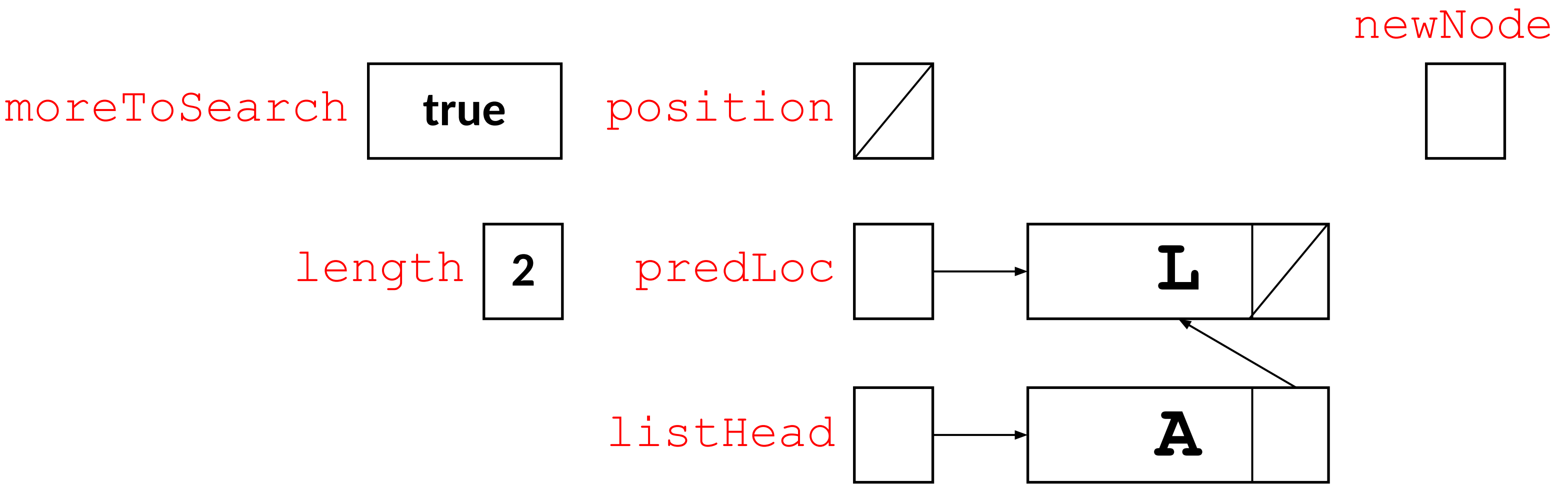
InsertItem('S')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



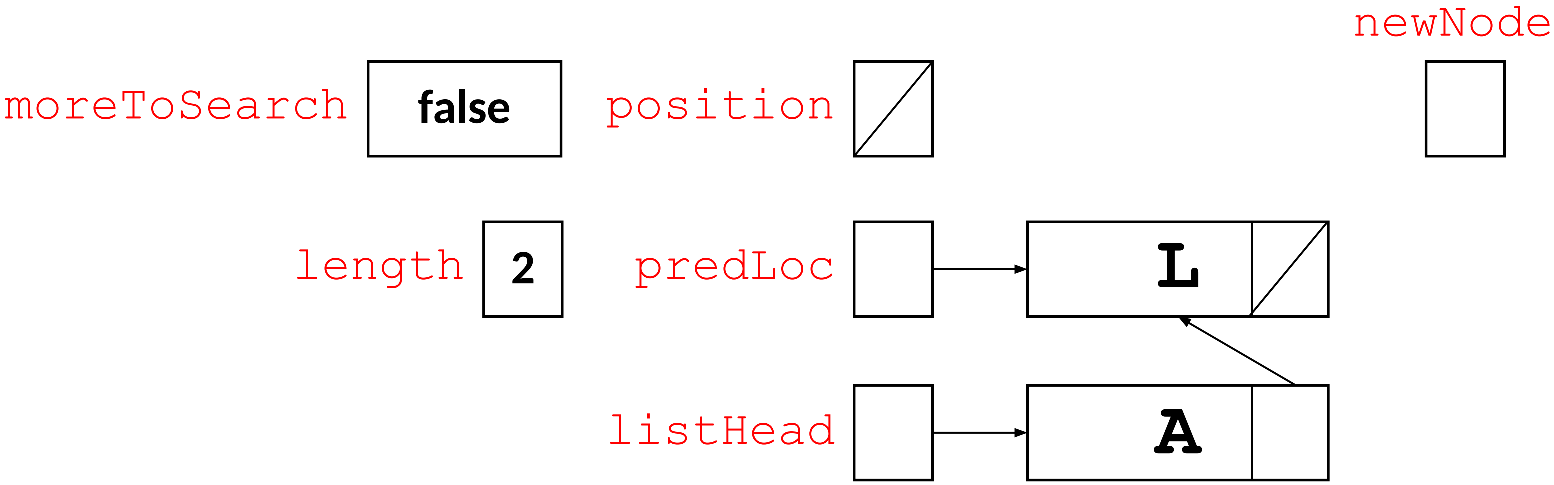
InsertItem('S')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



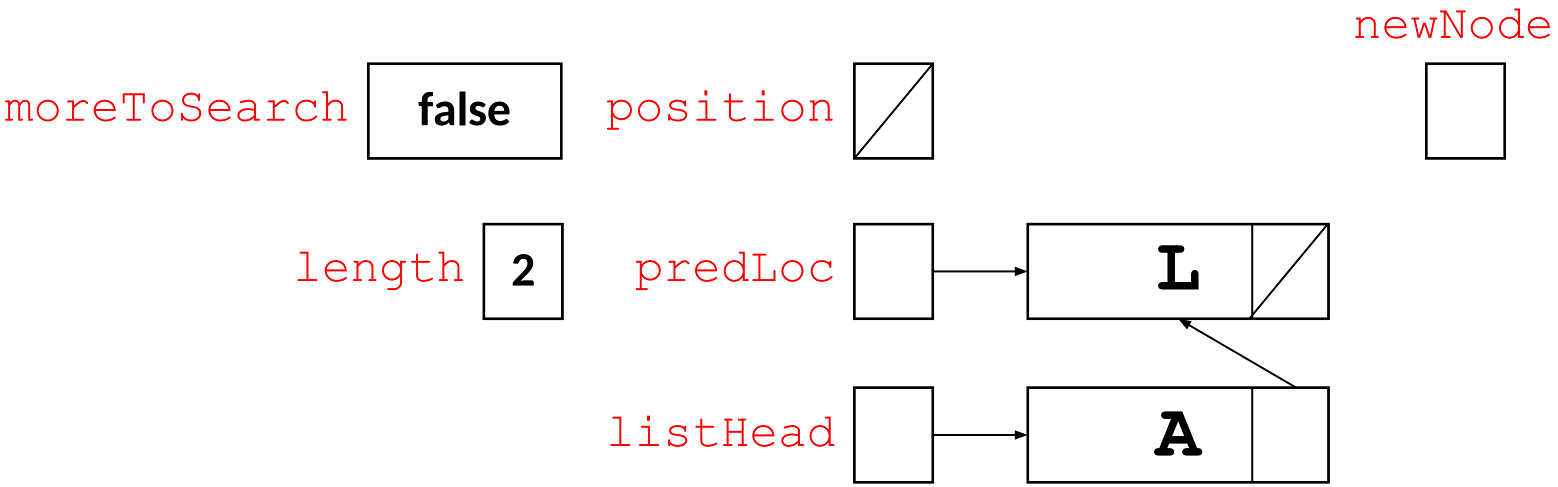
InsertItem('S')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



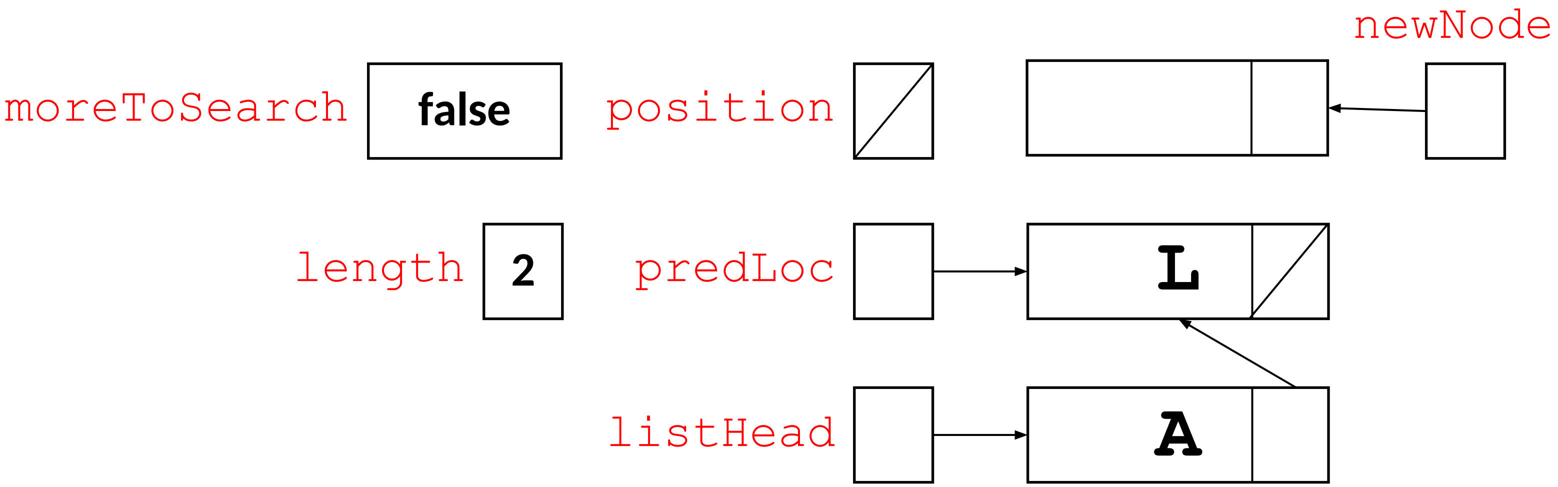
InsertItem('S')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



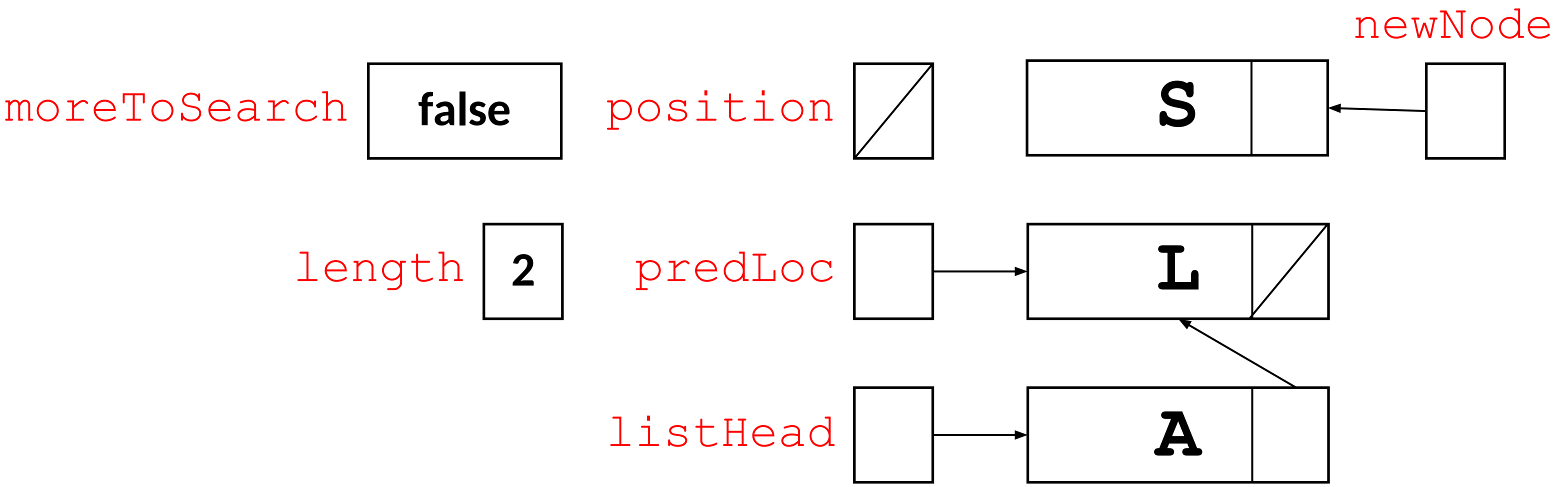
InsertItem('S')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



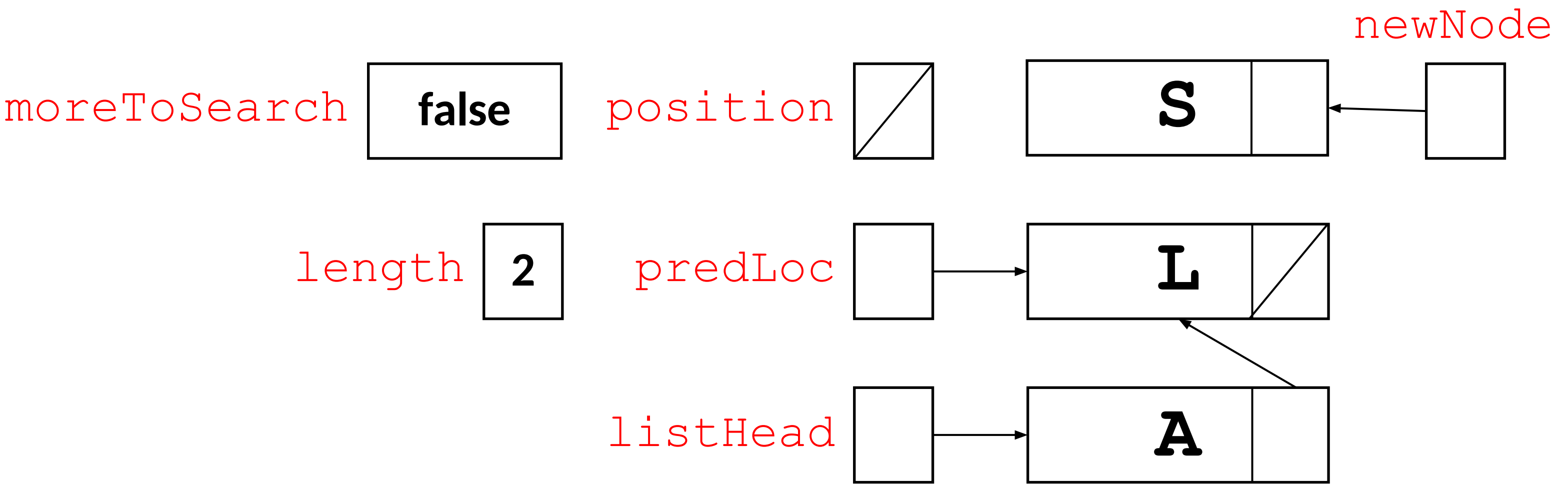
InsertItem('S')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



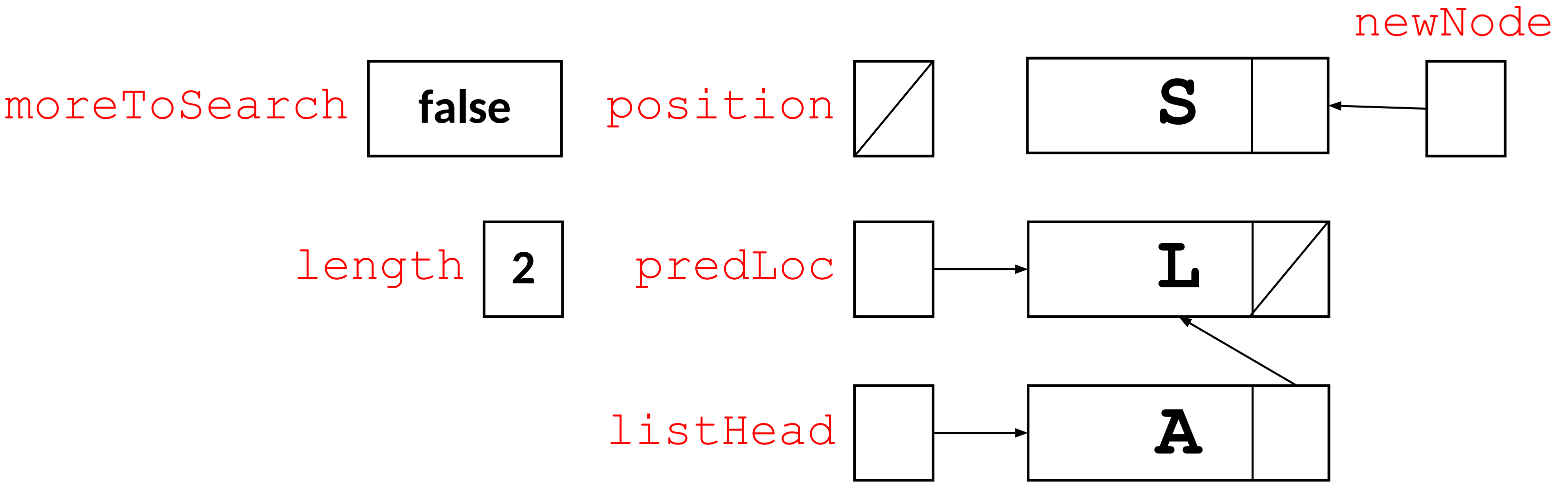
InsertItem('S')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



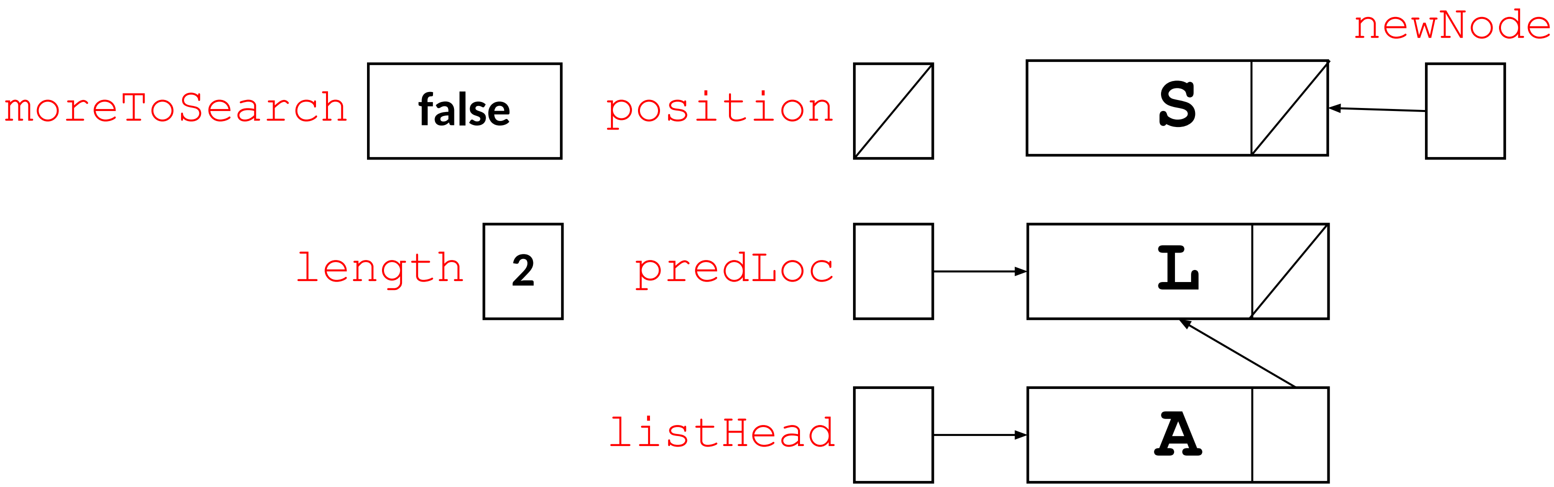
InsertItem('S')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



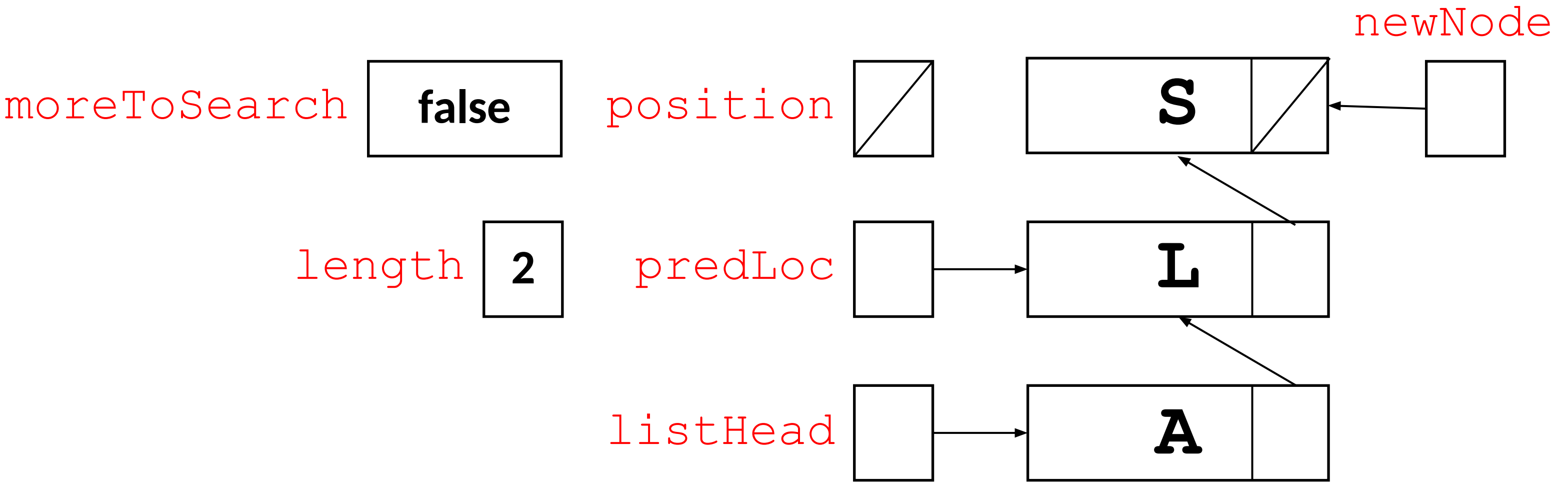
InsertItem('S')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



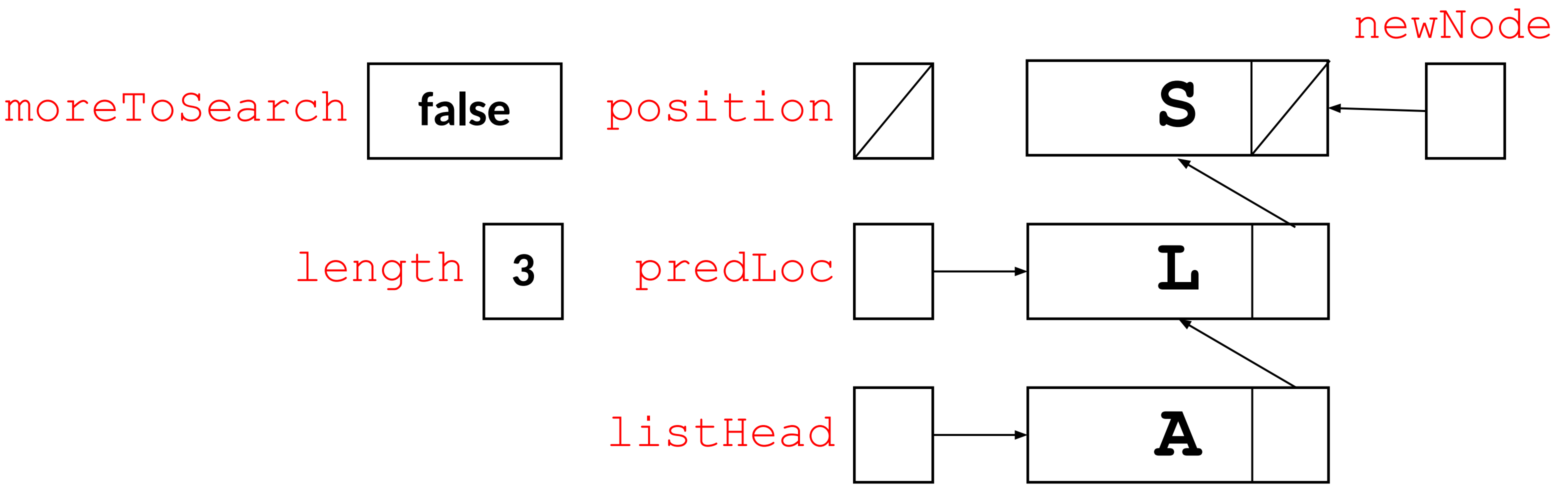
InsertItem('S')

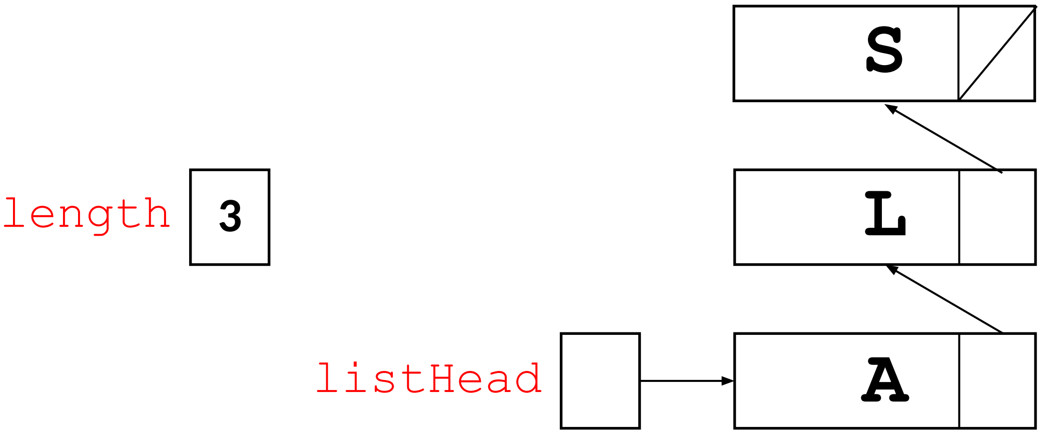
```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



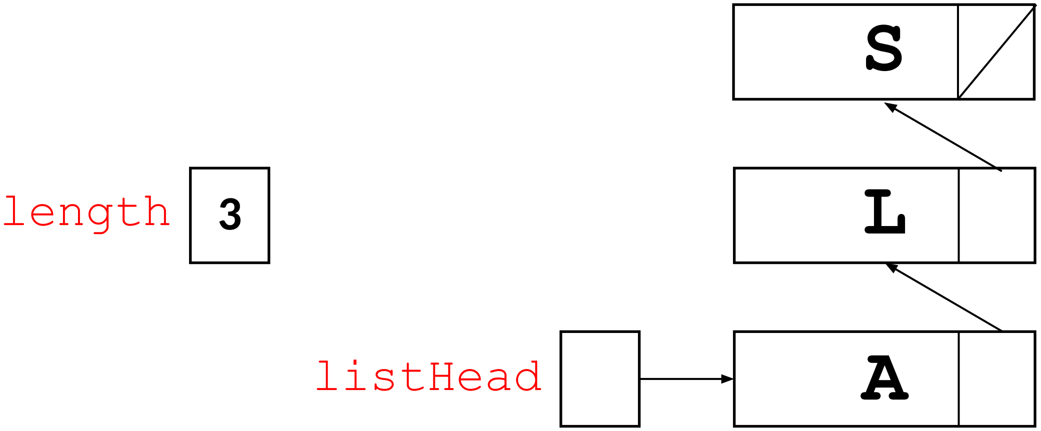
InsertItem('S')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



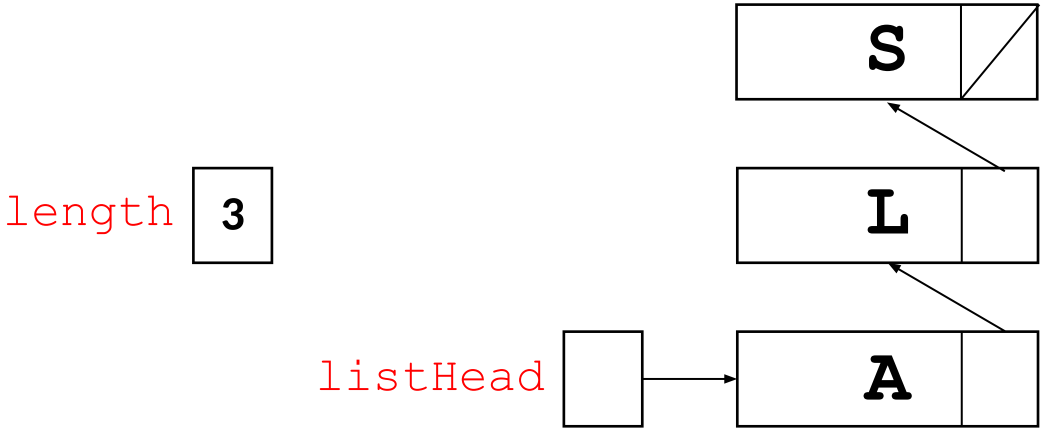

S
L
A
length
3
listHead

InsertItem('P')



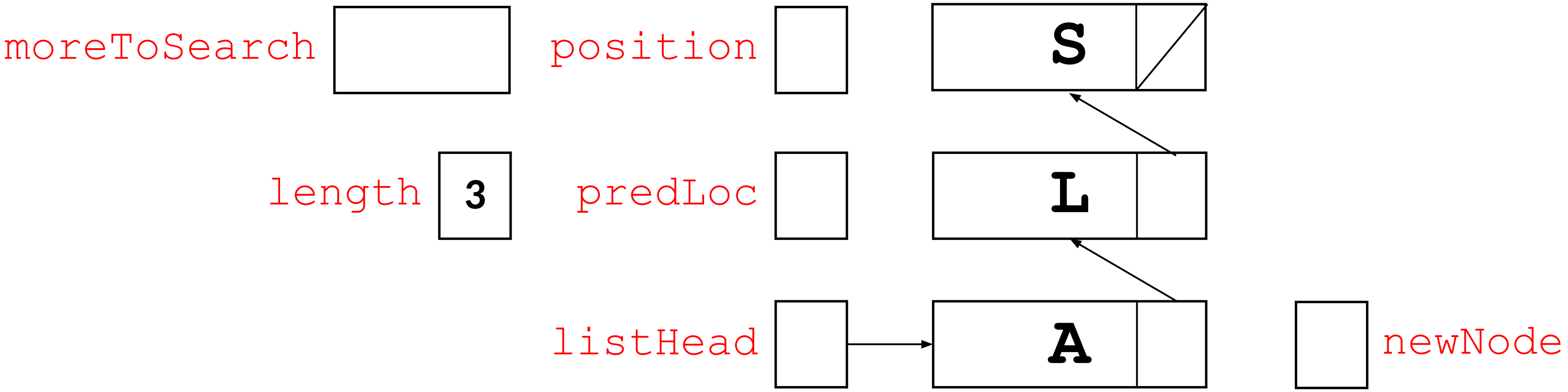
InsertItem('P')

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
```



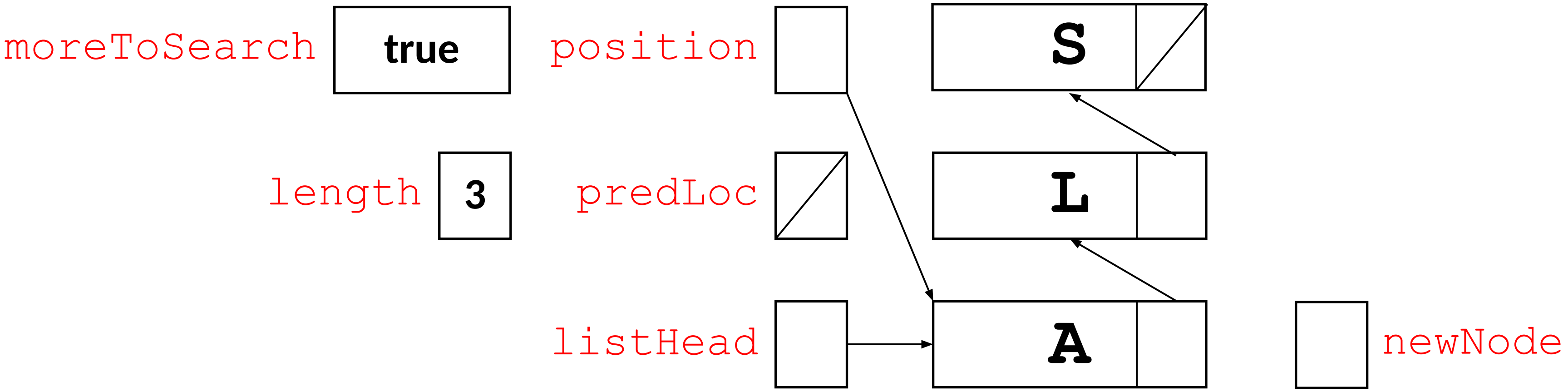
InsertItem('P')

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
```



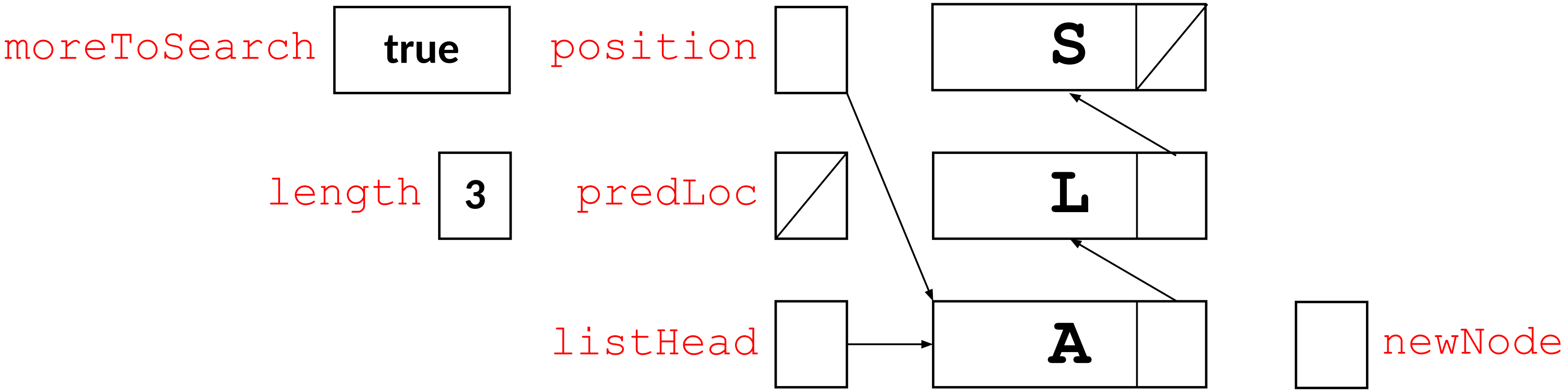
InsertItem('P')

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
```



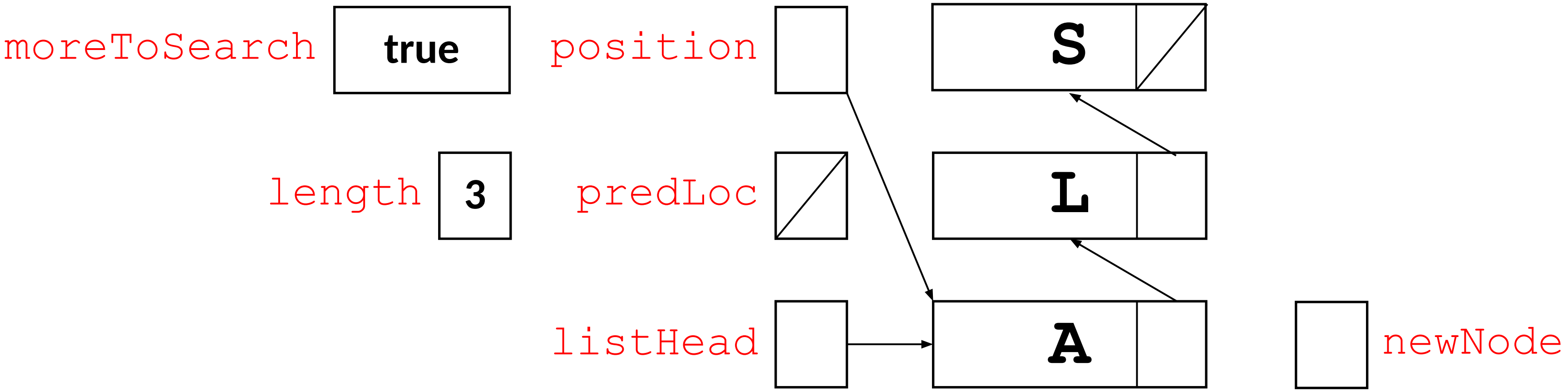
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



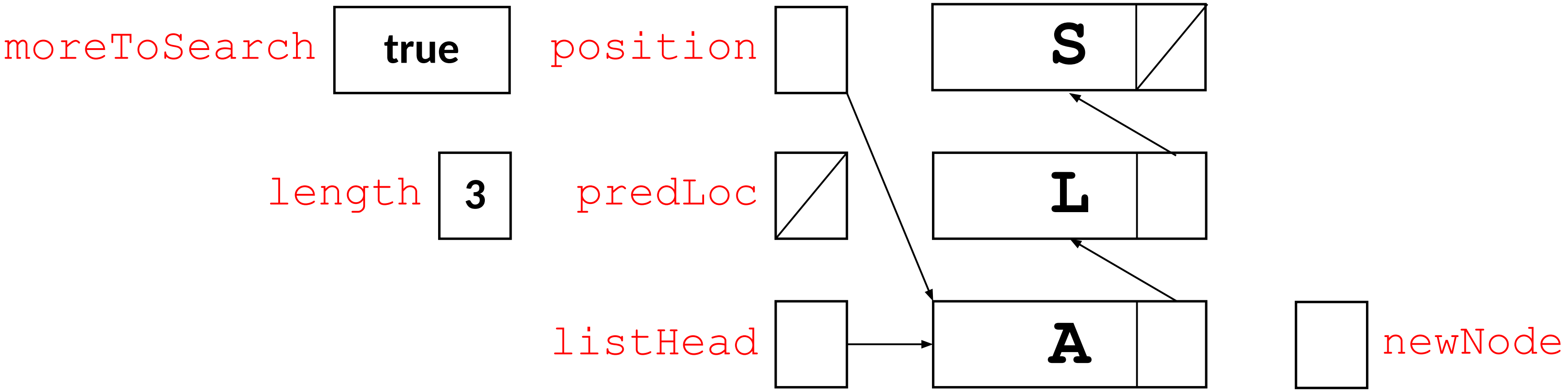
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



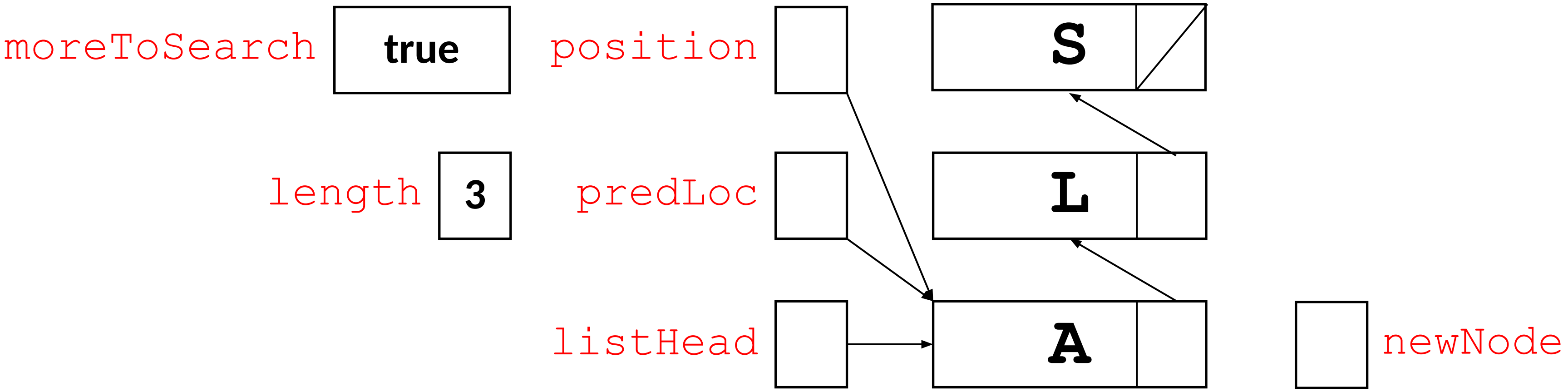
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



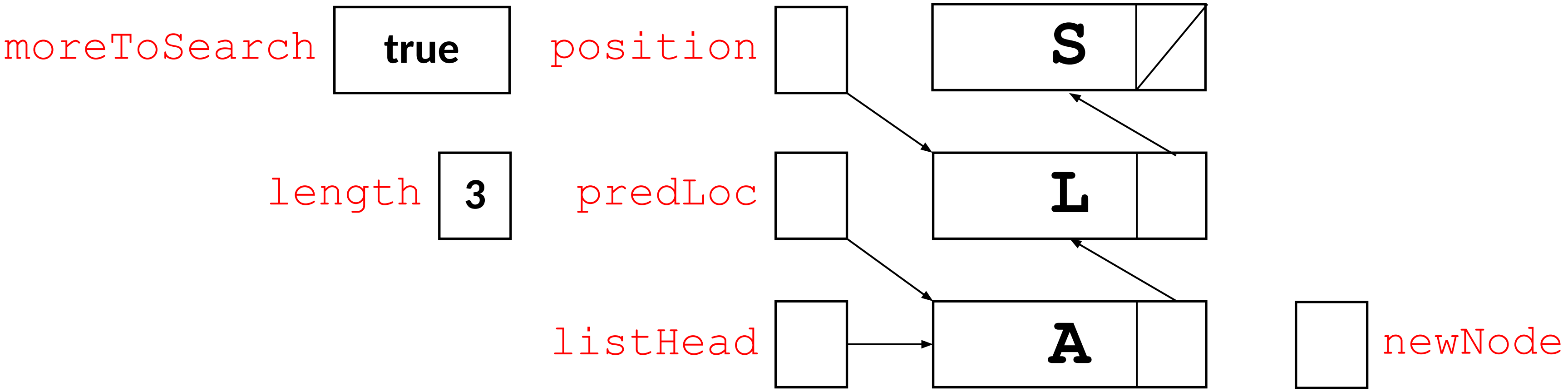
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



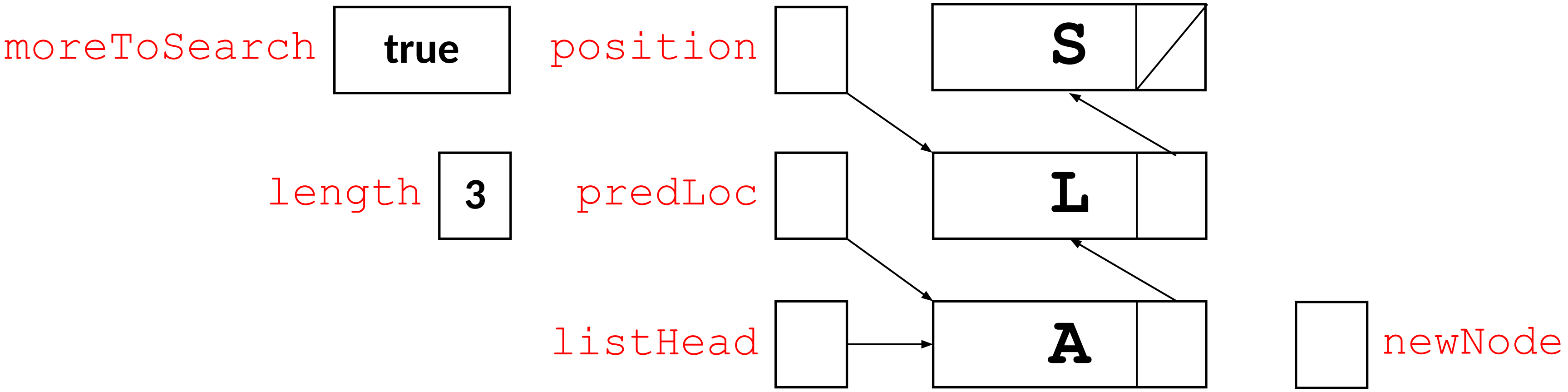
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



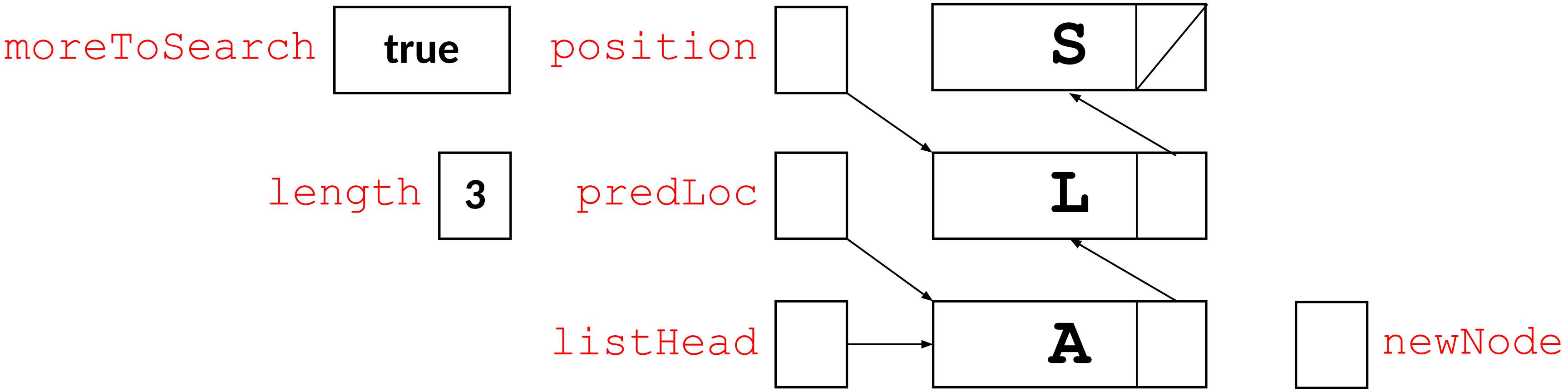
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



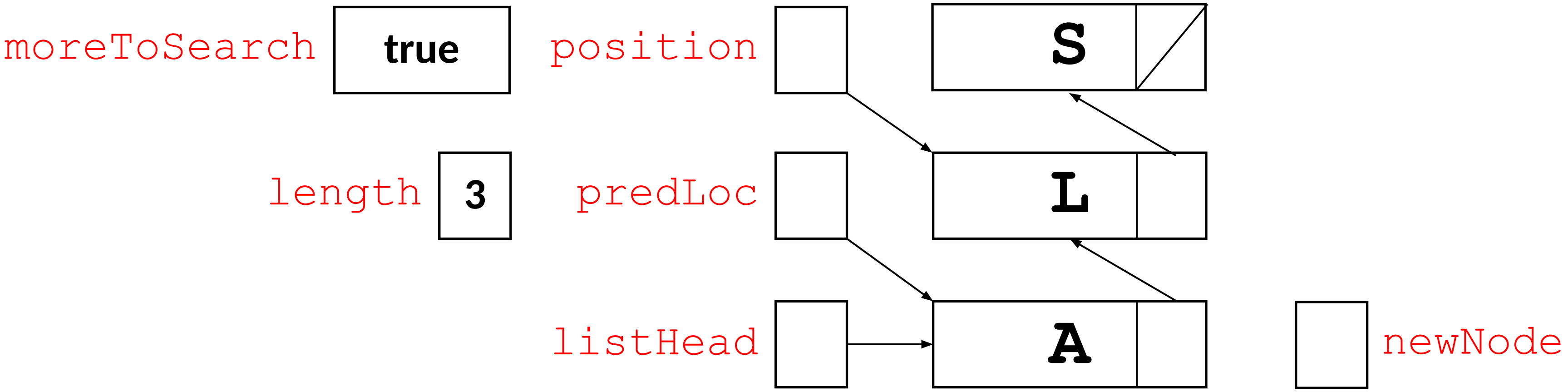
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



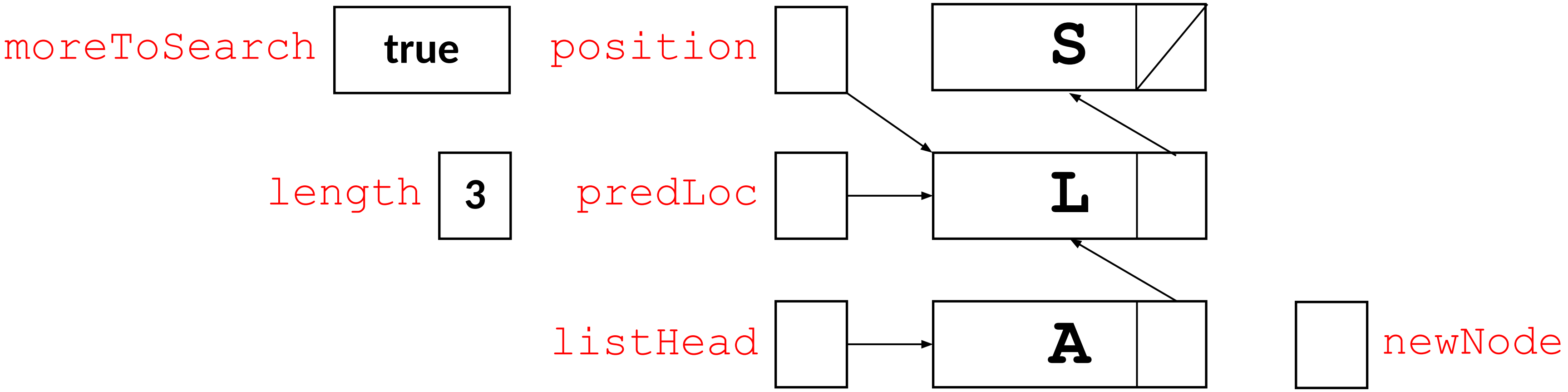
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



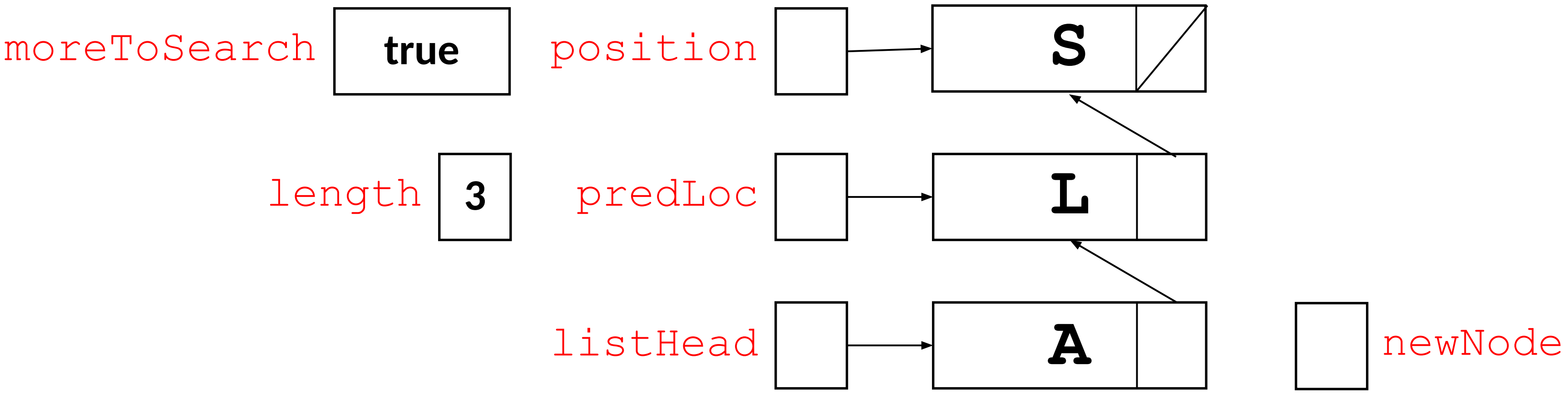
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



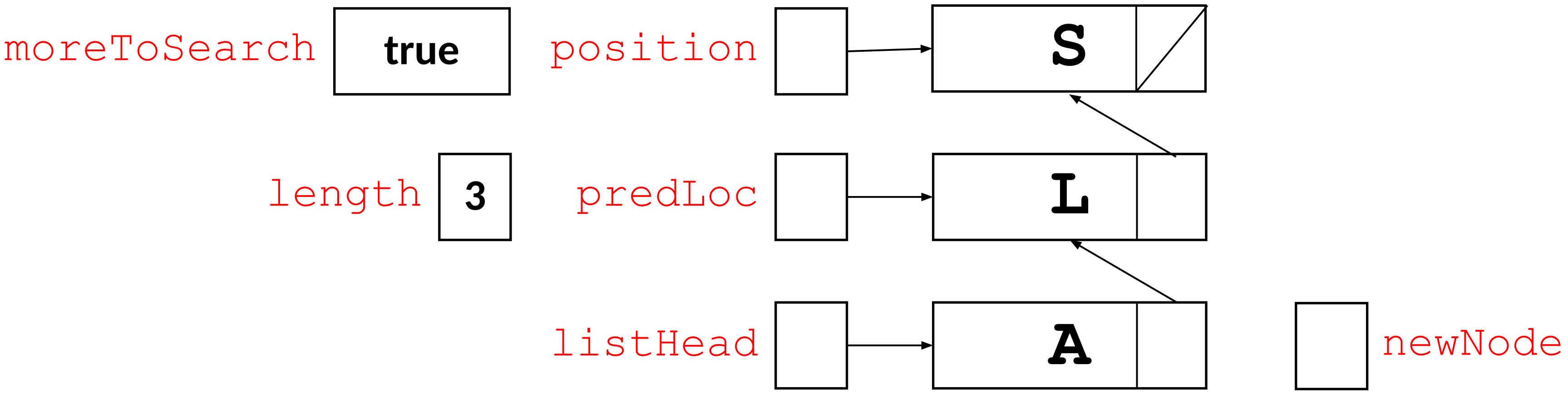
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



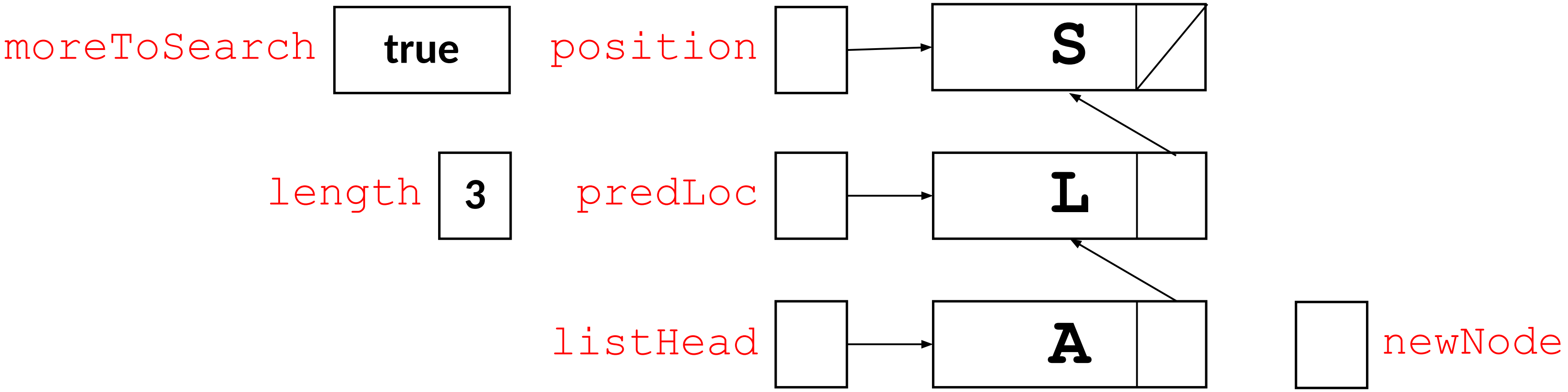
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



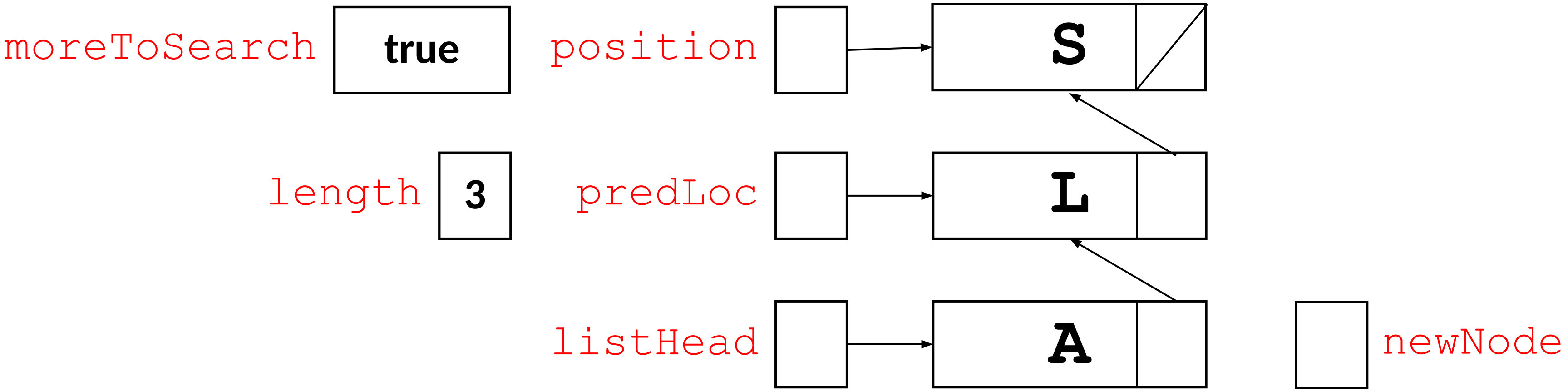
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



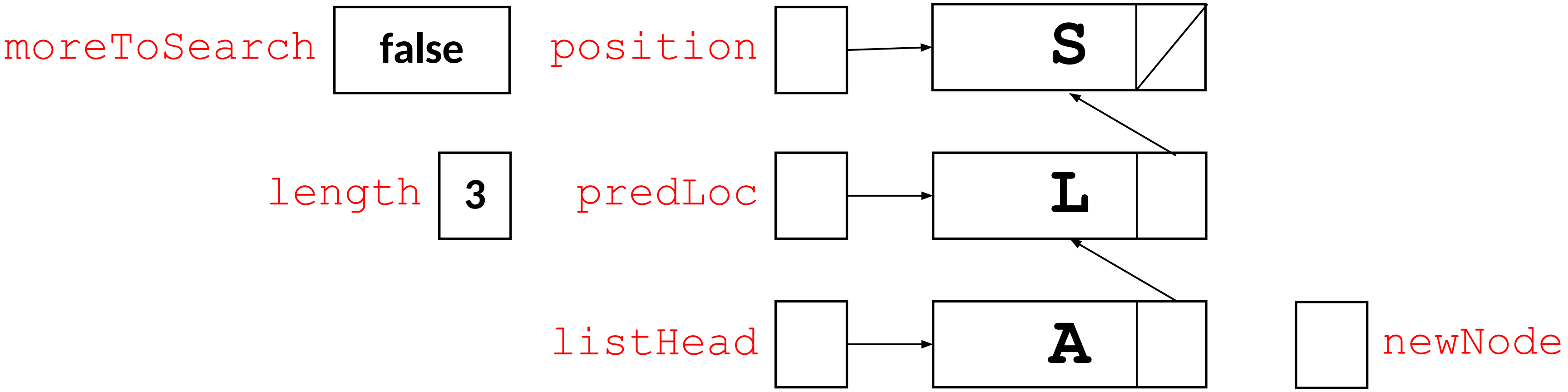
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



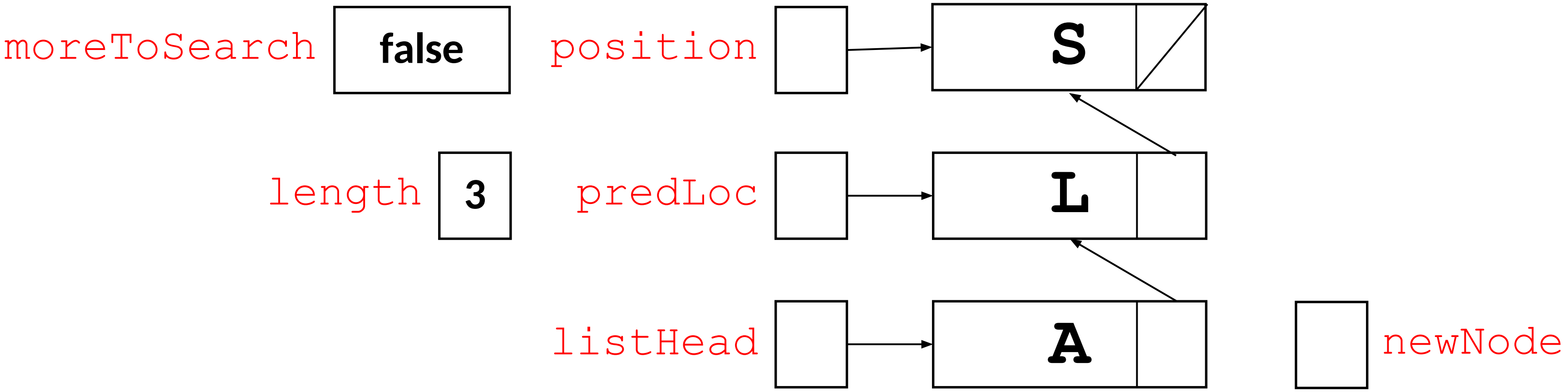
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



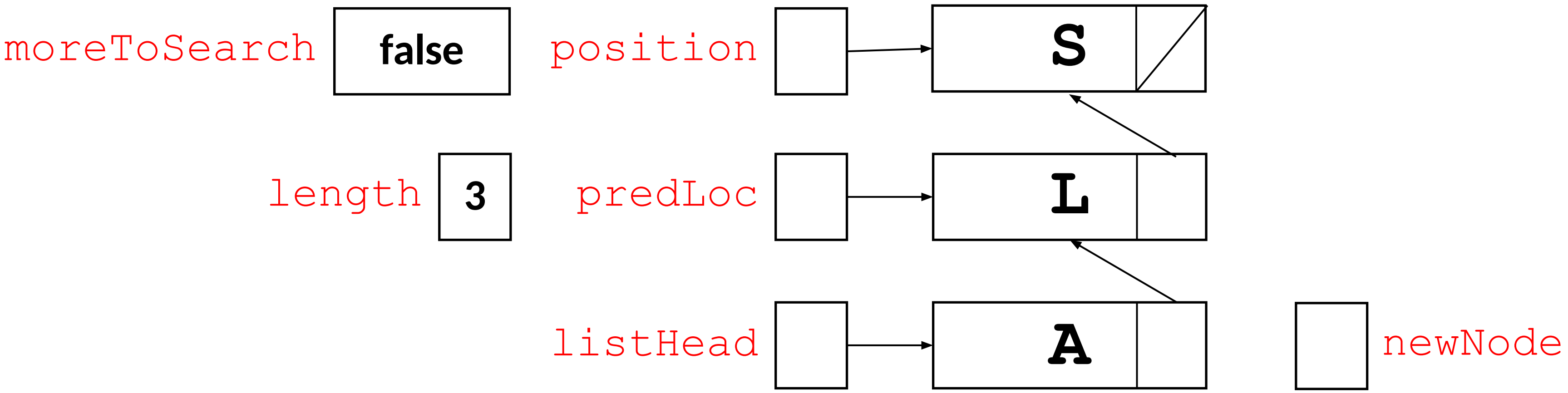
InsertItem('P')

```
while (moreToSearch)
 {
   if (position->info < item)
   {
     predLoc = position;
     position = position->next;
     moreToSearch = (position != NULL);
   }
   else moreToSearch = false;
 }
```



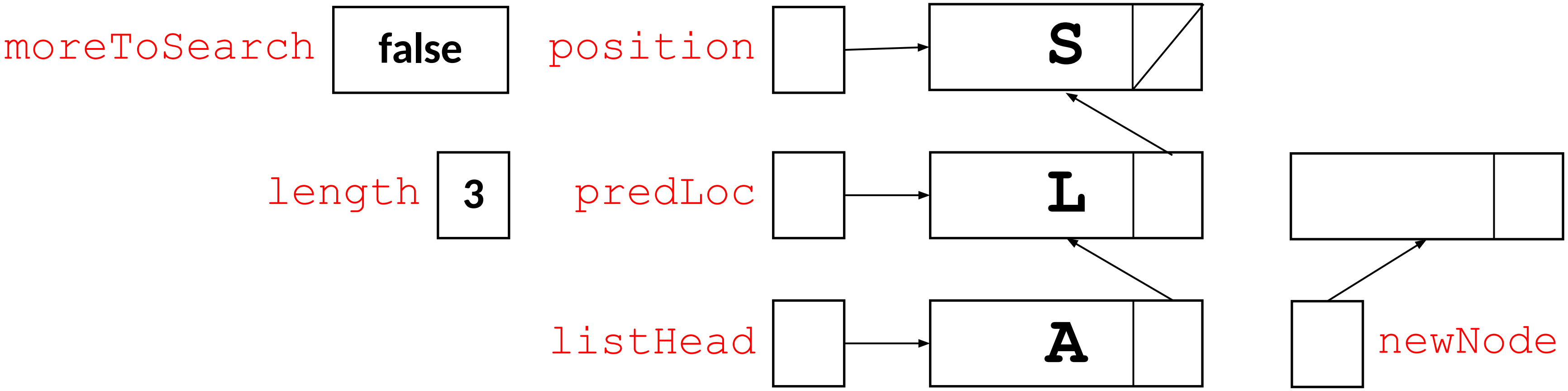
InsertItem('P')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



InsertItem('P')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



moreToSearch false

position

S

length 3

predLoc

L

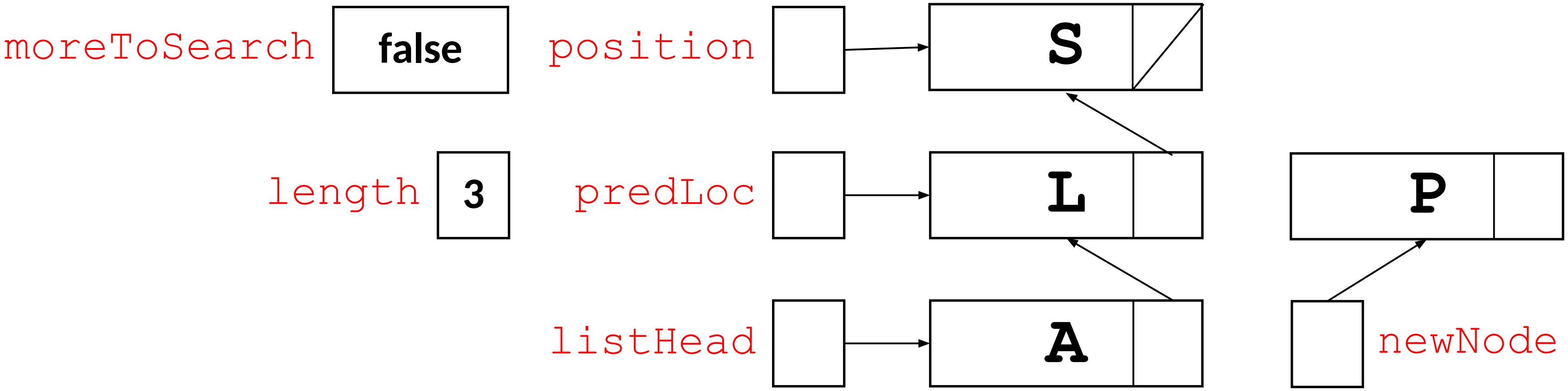
listHead

A

newNode

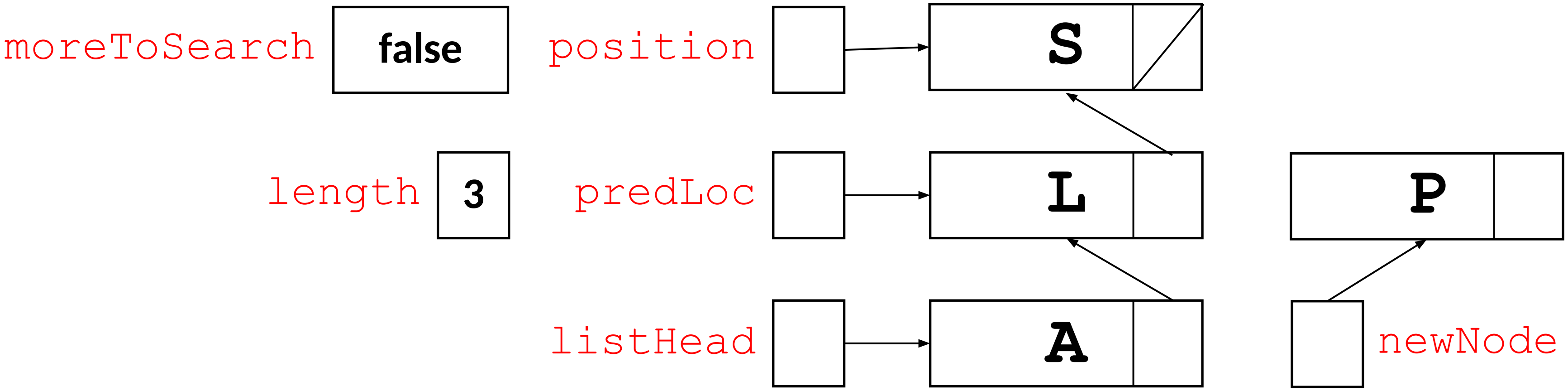
InsertItem('P')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



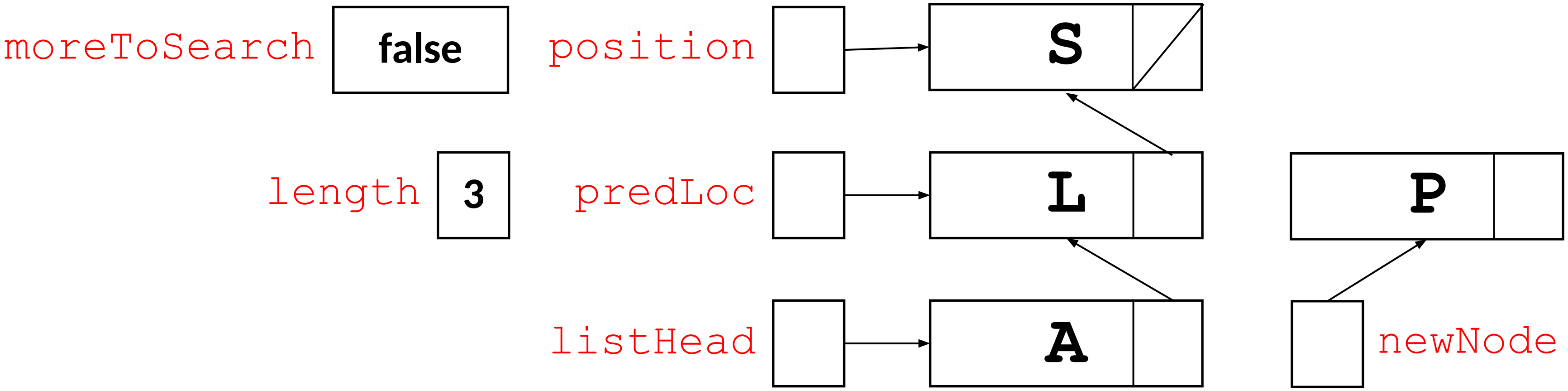
InsertItem('P')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



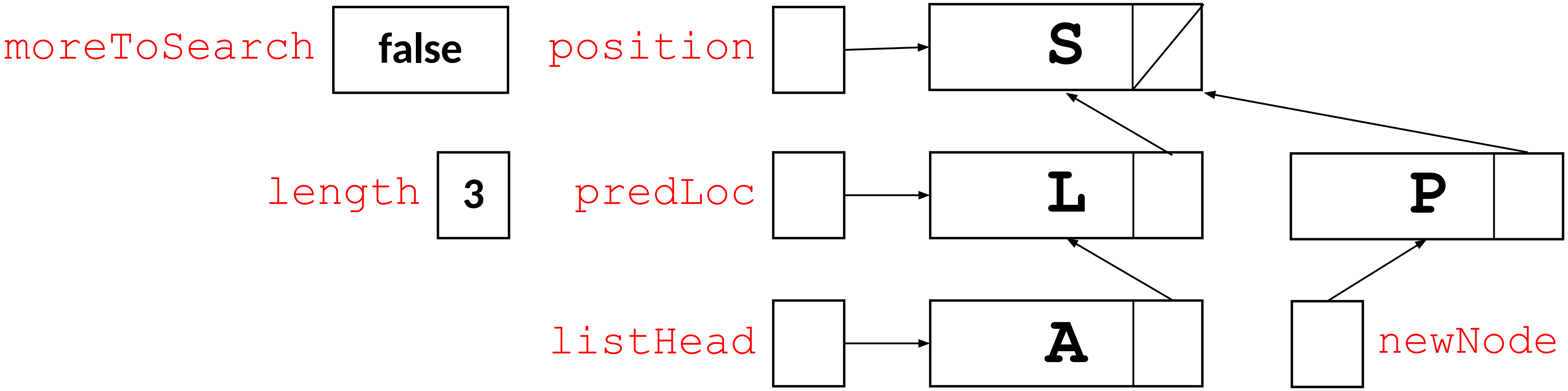
InsertItem(‘P’)

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



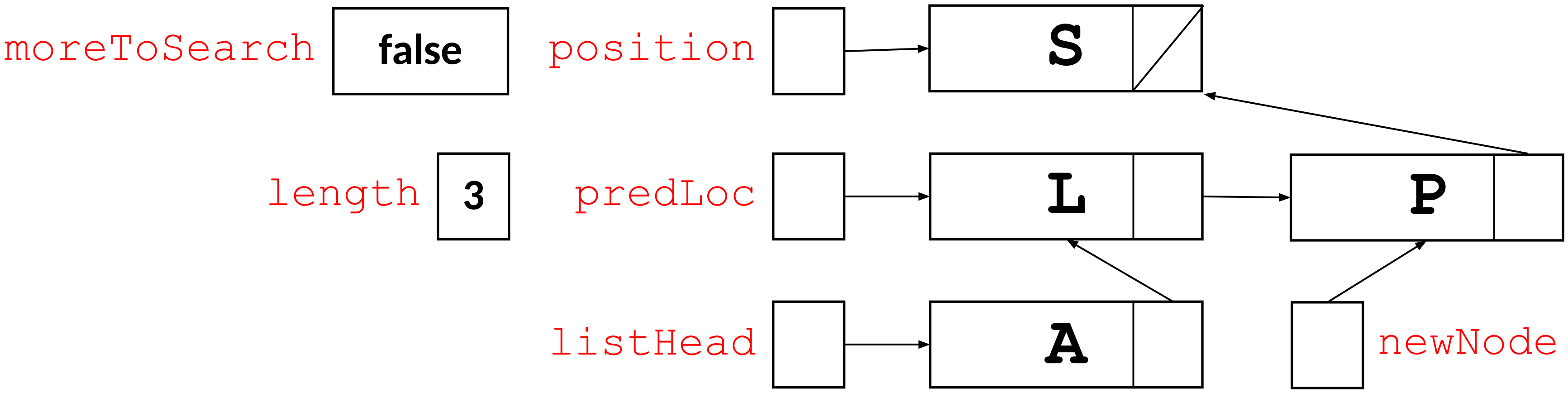
InsertItem('P')

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



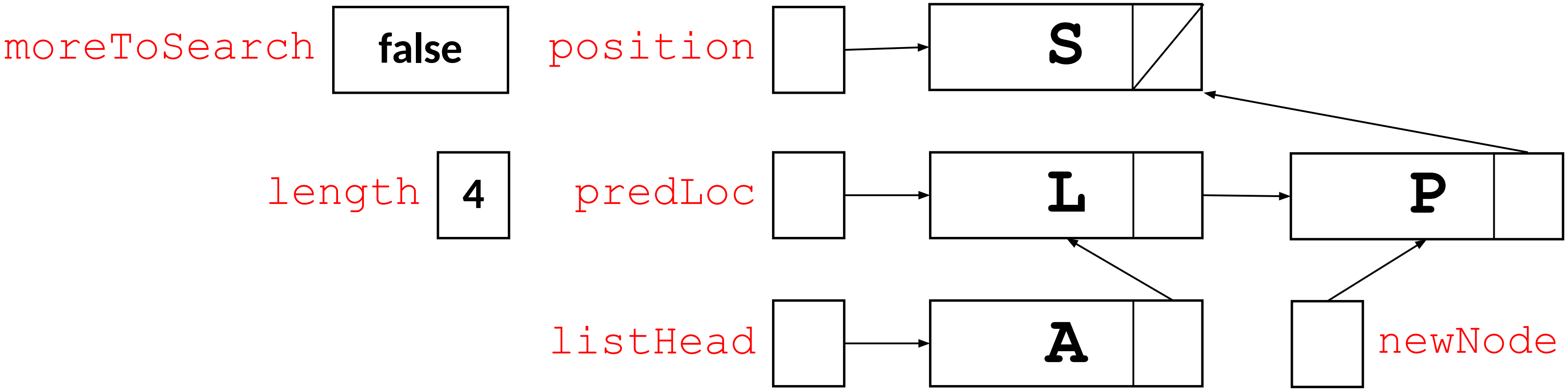
InsertItem('P')

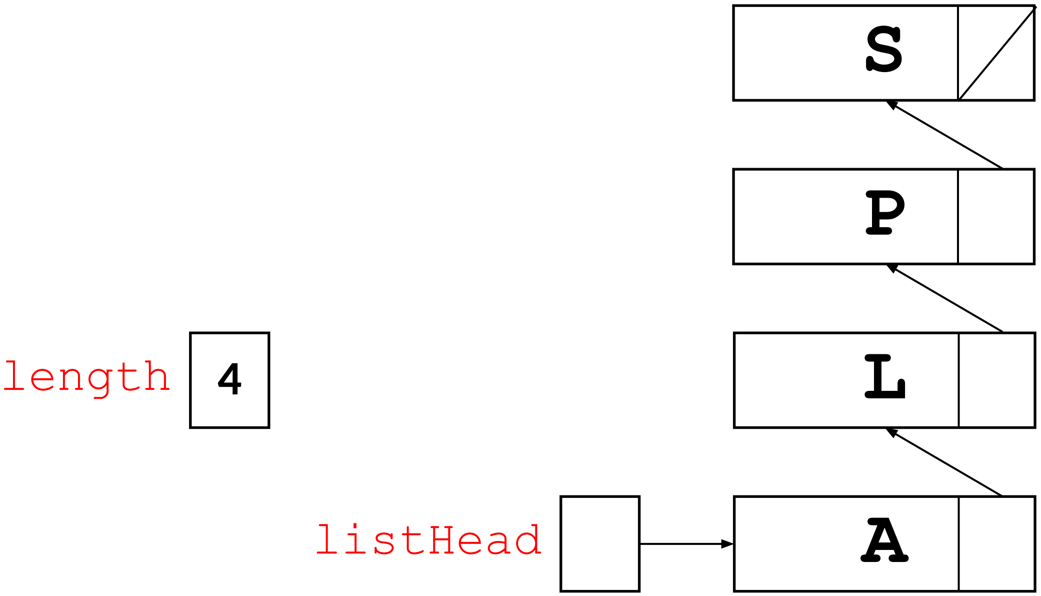
```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```



InsertItem(‘P’)

```
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```




S
P
length
4
L
listHead
A

sortedlinkedlist.cpp

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
  NodeType* newNode;
  NodeType* predLoc;
  NodeType* position;
  bool moreToSearch;

  position = listHead;
  predLoc = NULL;
  moreToSearch = (position != NULL);
  while (moreToSearch)
  {
    if (position->info < item)
    {
      predLoc = position;
      position = position->next;
      moreToSearch = (position != NULL);
    }
    else moreToSearch = false;
  }
  newNode = new NodeType;
  newNode->info = item;

  if (predLoc == NULL)
  {
    newNode->next = listHead;
    listHead = newNode;
  }
  else
  {
    newNode->next = position;
    predLoc->next = newNode;
  }
  length++;
}
```

O(N)

sortedlinkedlist.cpp

```
template <class ItemType>
void SortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* position = listHead;
  NodeType* tempPtr;
  if (item == listHead->info)
  {
    tempPtr = position;
    listHead = listHead->next;
  }
  else
  {
    while (!(item==(position->next)->info))
      position = position->next;
    tempPtr = position->next;
    position->next = (position->next)->next;
  }
  delete tempPtr;
  length--;
}
```

sortedlinkedlist.cpp

```
template <class ItemType>
void SortedType<ItemType>::DeleteItem(ItemType item)
{
  NodeType* position = listHead;
  NodeType* tempPtr;
  if (item == listHead->info)
  {
    tempPtr = position;
    listHead = listHead->next;
  }
  else
  {
    while (!(item==(position->next)->info))
      position = position->next;
    tempPtr = position->next;
    position->next = (position->next)->next;
  }
  delete tempPtr;
  length--;
}
```

O(N)

sortedlinkedlist.cpp

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

sortedlinkedlist.cpp

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

O(N)