COMPREHENSIVE EDITION

PROGRAMMING AND PROBLEM SOLVING WITH C++

SIXTH EDITION

Nell Dale and Chip Weems

Chapter 13

Applied Arrays: Lists and Strings

Background image © Toncsi/ShutterStock, Inc.
Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Chapter 13 Topics

- Meaning of a List
- Insertion and Deletion of List Elements
- Selection Sort of List Elements
- Insertion and Deletion using a Sorted List
- Binary Search in a Sorted List
- Order of Magnitude of a Function
- Declaring and Using C Strings
- Using `typedef` with Arrays

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Chapter 13 Topics

- Meaning of a List
- Insertion and Deletion of List Elements
- Selection Sort of List Elements
- Insertion and Deletion using a Sorted List

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Chapter 13 Topics

- Binary Search in a Sorted List
- Order of Magnitude of a Function
- Declaring and Using C Strings
- Using `typedef` with Arrays

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

What is a List?

- A **list** is a variable-length, linear collection of homogeneous elements
- **Linear** means that each list element (except the first) has a unique predecessor, and each element (except the last) has a unique successor

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

4 Basic Kinds of ADT Operations

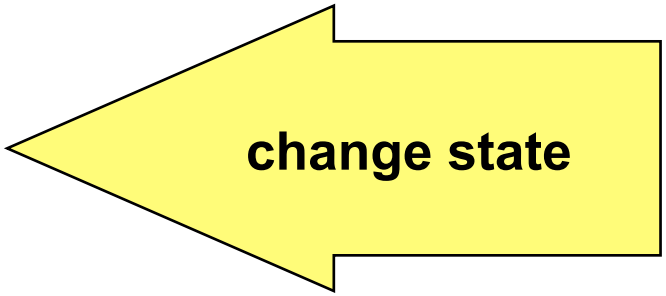
- **Constructors** -- create a new instance (object) of an ADT
- **Transformers** -- change the state of one or more of the data values of an instance
- **Observers** -- allow client to observe the state of one or more of the data values of an instance without changing them
- **Iterators** -- allow client to access the data values in sequence

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

ADT List Operations

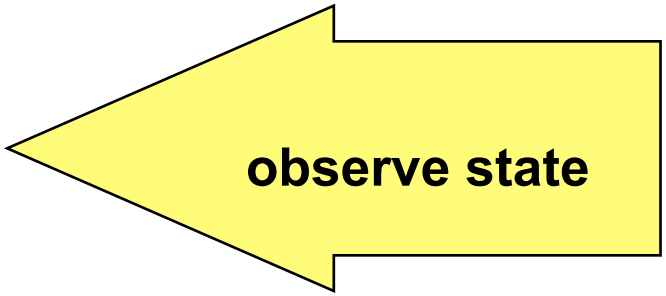
Transformers

- Insert
- Delete
- Sort



Observers

- IsEmpty
- IsFull
- Length
- IsPresent



Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

ADT List Operations

Iterator

- Reset
- GetNextItem

- Reset prepares for the iteration
- GetNextItem returns the next item in sequence
- No transformer can be called between calls to GetNextItem *(Why?)*

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

ADT Unsorted List Data Components

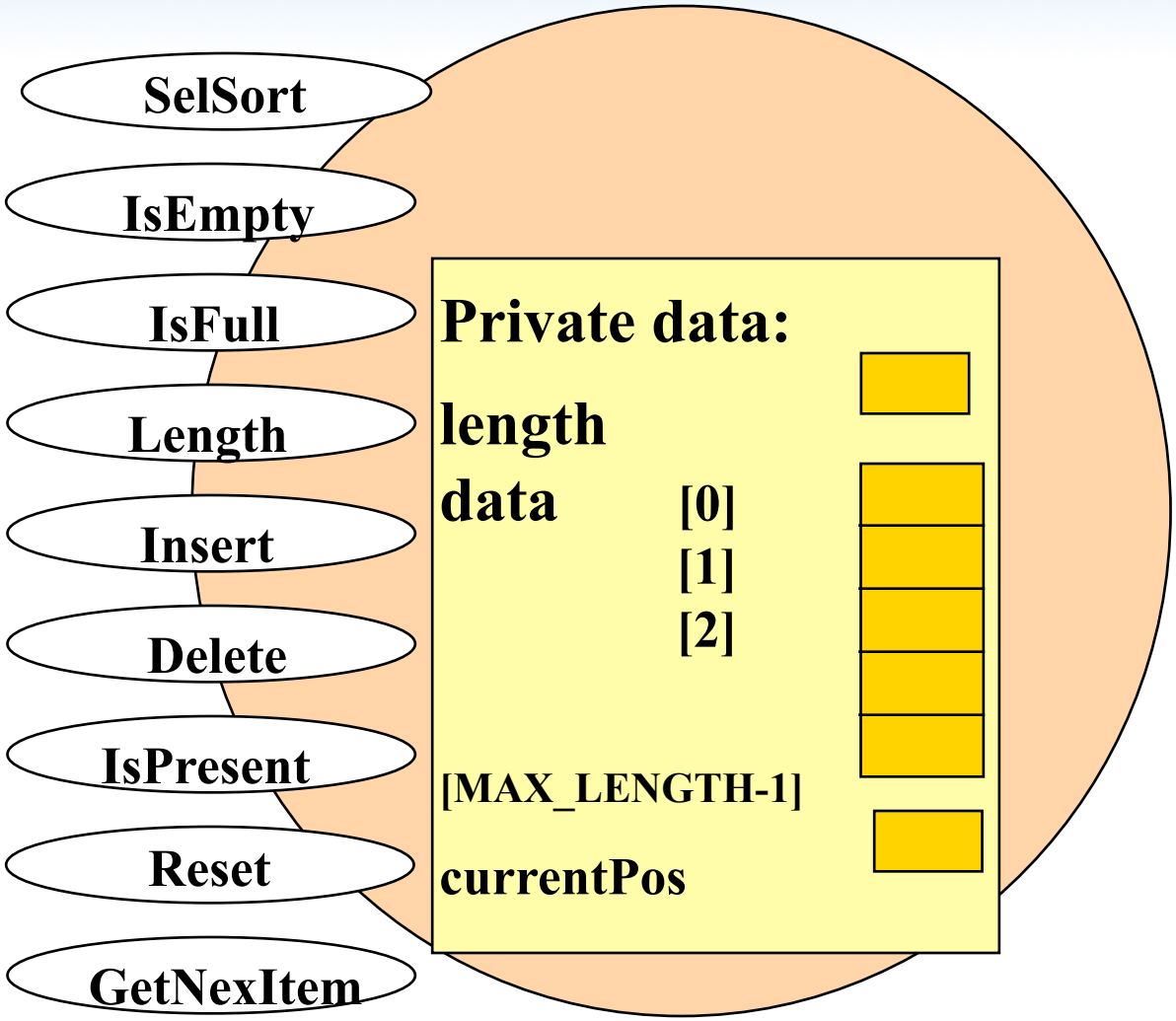
length — number of elements in list

`data[0.. MAX_LENGTH -1]` — array of list elements

`currentPos` — used in iteration

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Array-based class List



Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
// Specification file array-based list ("list.h")
const  int  MAX_LENGTH  =  50;
typedef int   ItemType;

class  List                // Declares a class data type
{
public:                    // Public member functions

    List();                // constructor
    bool IsEmpty () const;
    bool IsFull ()  const;
    int  Length ()  const; // Returns length of list
    void Insert (ItemType  item);
    void Delete (ItemType  item);
    bool IsPresent(ItemType  item)  const;
    void SelSort ();
    void Reset ();
    ItemType GetNextItem ();
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
private:            // Private data members
    int length; // Number of values currently stored
    ItemType data[MAX_LENGTH];
    int  CurrentPos;  // Used in iteration
};
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Sorted and Unsorted Lists

UNSORTED LIST	SORTED LIST
Elements are placed into the list in no particular order	List elements are in sorted in some way -- either numerically or alphabetically

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
// Implementation file array-based list
// ("list.cpp")

#include  "list.h"
#include  <iostream>

using namespace std;

int  List::Length ()  const
// Post: Return value is length
{
    return  length;
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

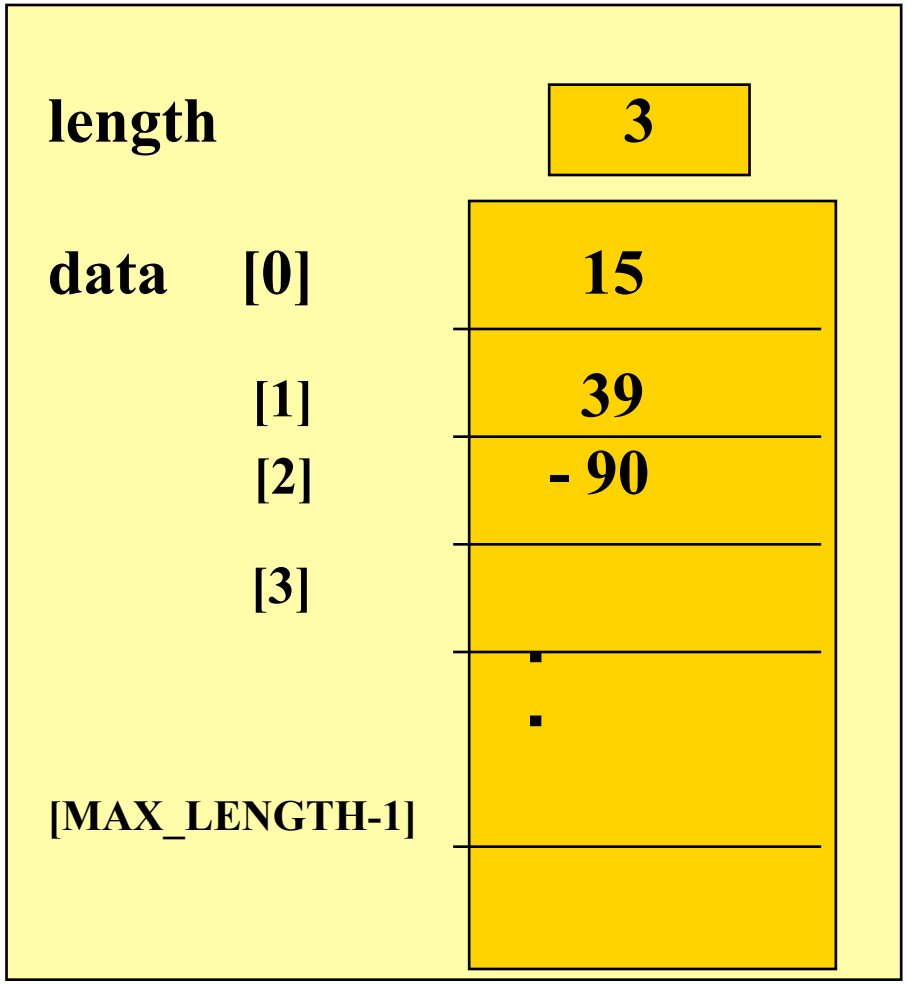
```
bool  List::IsFull ()  const
// Post: Return value is true
//       if length is equal
//  to MAX_LENGTH and false otherwise
{
    return (length == MAX_LENGTH);
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
List::List ()
// Constructor
// Post: length == 0
{
    length = 0;
}

void  List::Insert (/* in */  ItemType  item)
// Pre: length < MAX_LENGTH && item is assigned
// Post: data[length@entry] == item &&
//       length == length@entry + 1
{
    data[length] = item;
    length++;
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Before Inserting 64 into an Unsorted List

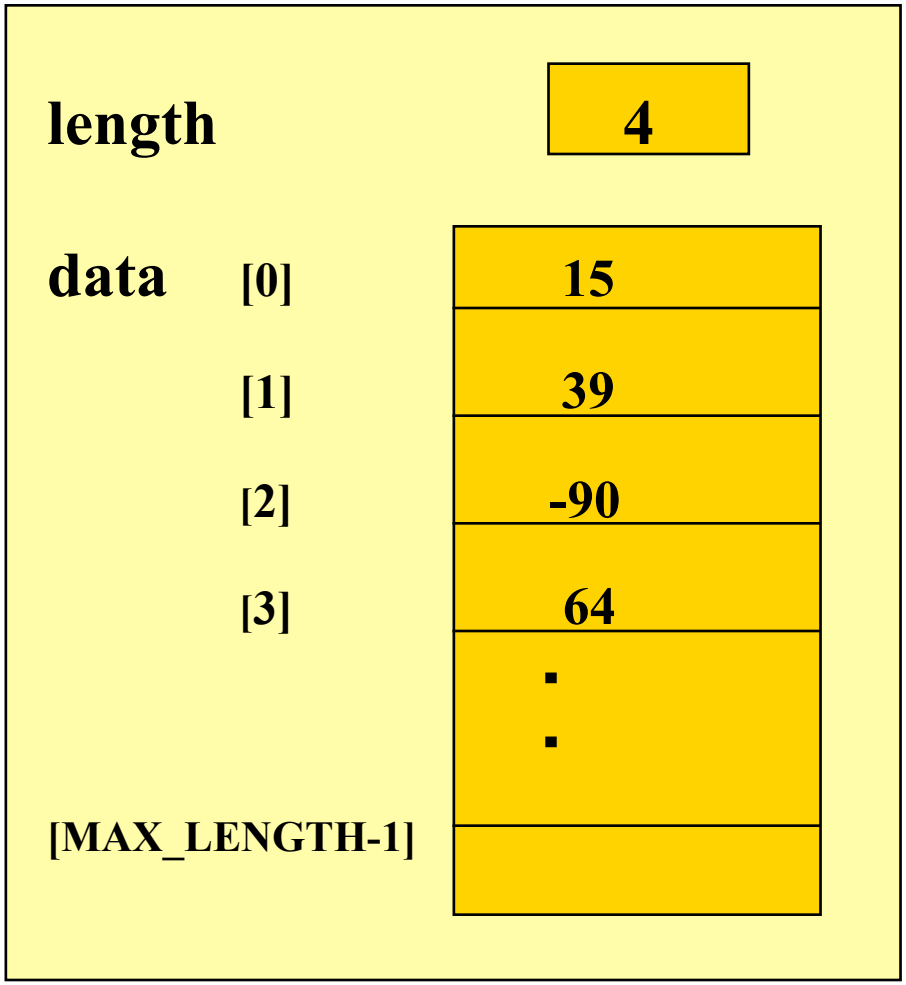


The item will be placed into the length location, and length will be incremented

item 64

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

After Inserting 64 into an Unsorted List



The item will be placed into the length location, and length will be incremented

item 64

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
bool  List::IsEmpty ()  const
// Post: Return value is true if length is equal
//  to zero and false otherwise
{
    return (length == 0);
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
bool List::IsPresent( /* in */ ItemType item)
  const
// Searches the list for item, reporting
//   whether found
// Post: Function value is true, if item is in
//   data[0 . . length-1] and is false otherwise
{
    int index  =  0;
    while (index < length && item != data[index])
        Index++;
    return  (index < length);
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

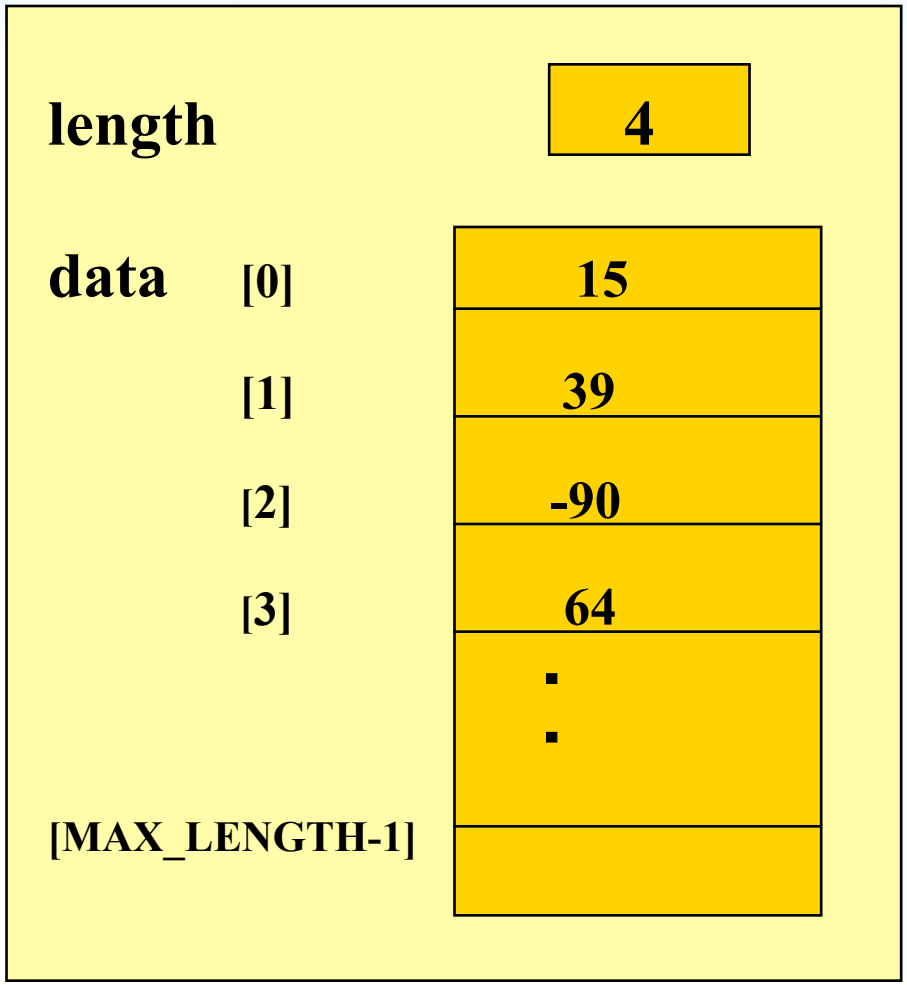
```
void  List::Delete ( /* in */  ItemType  item)
// Pre: length > 0  &&  item is assigned
// Post: IF item is in data array at entry
//    First occurrence of item is no longer
//    in array
//       && length == length@entry - 1
//    ELSE
//        length and data array are unchanged
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
{
    int  index  =  0;

    while (index < length  &&
           item != data[index])
        index++;
    // IF item found, move last element into
    //  item's place
    if (index < length)
    {
        data[index] = data[length - 1];
        length--;
    }
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Deleting 39 from an Unsorted List



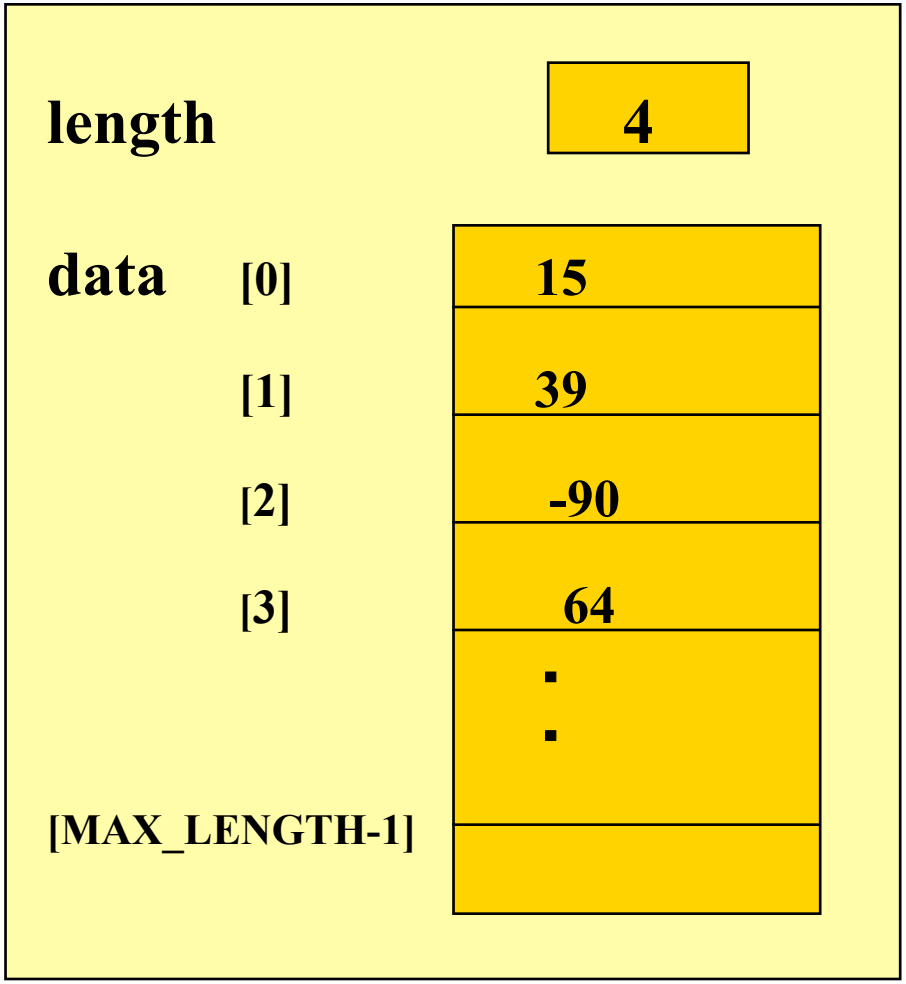
index: 0

39 has not been matched

item 39

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Deleting 39 from an Unsorted List



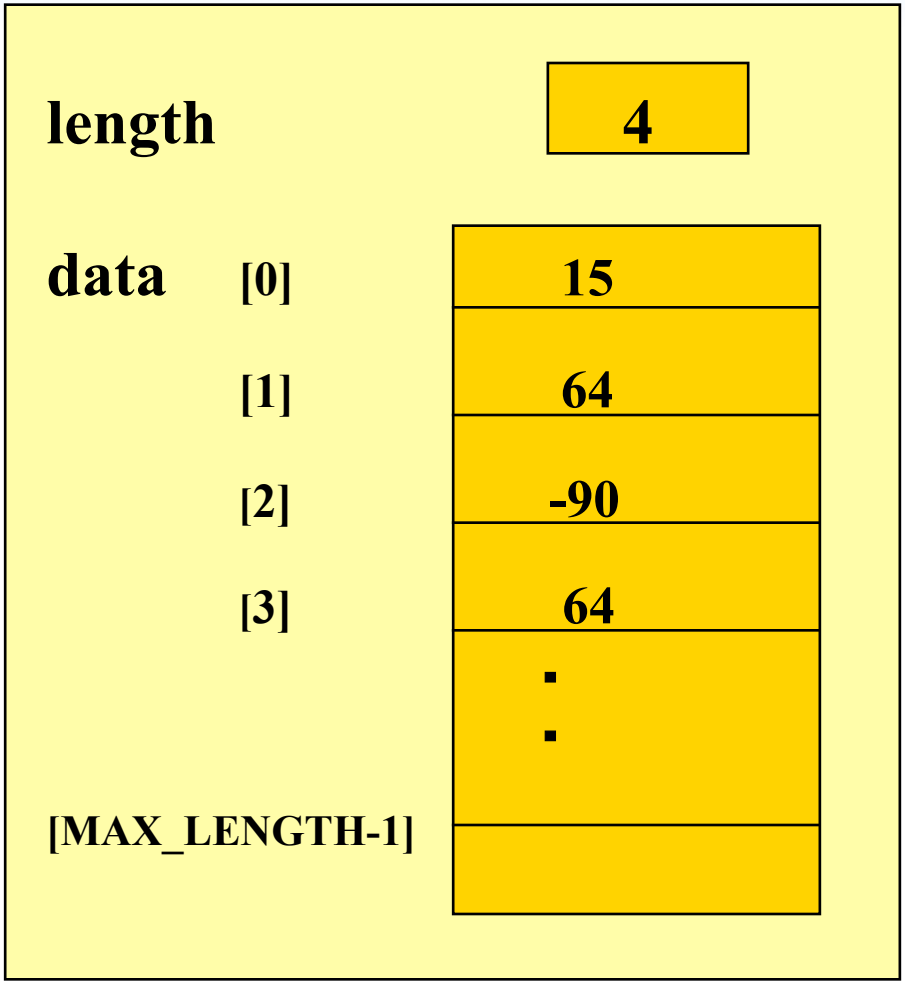
index: 1

39 has been matched

item 39

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Deleting 39 from an Unsorted List



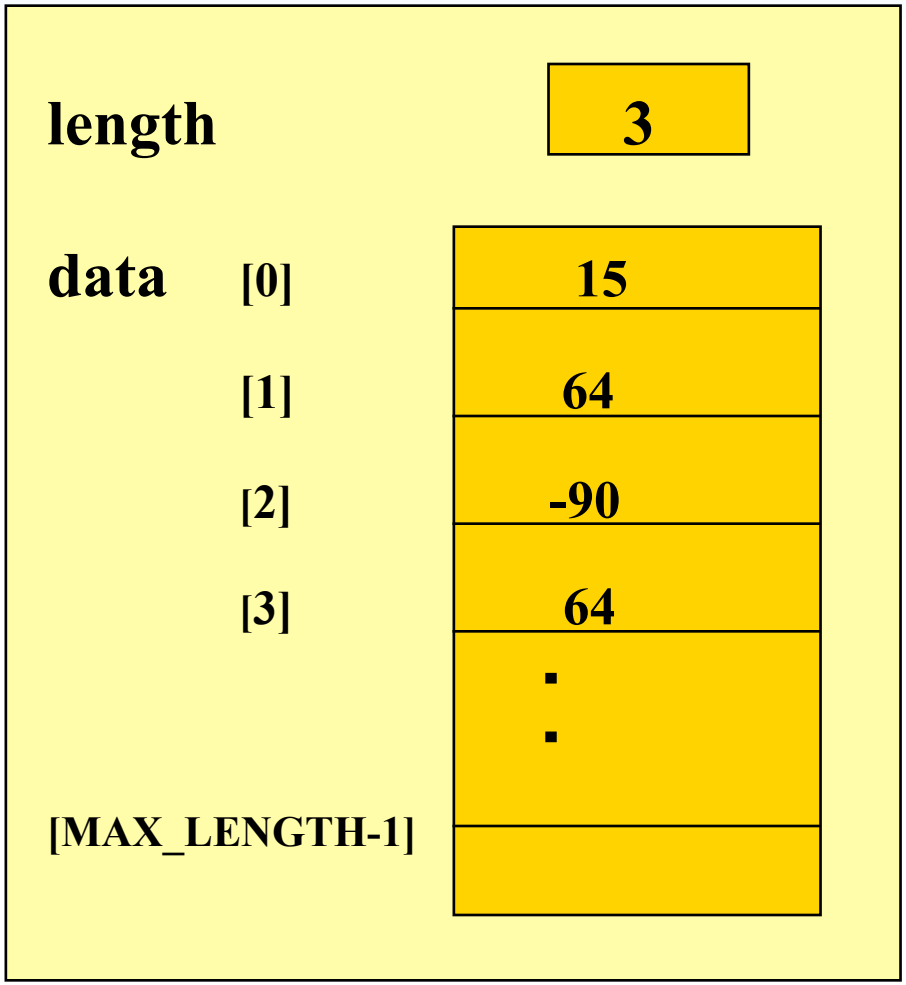
index: 1

Placed copy of last list element into the position where 39 was before

item 39

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Deleting 39 from an Unsorted List



index: 1

Decremented length

item 39

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Preparing for Iteration

What should currentPos be initialized to in order to access the first item?

```
void List::Reset()
// Post: currentPos has been initialized.
 {
     currentPos = 0;
 }
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Iteration Operator

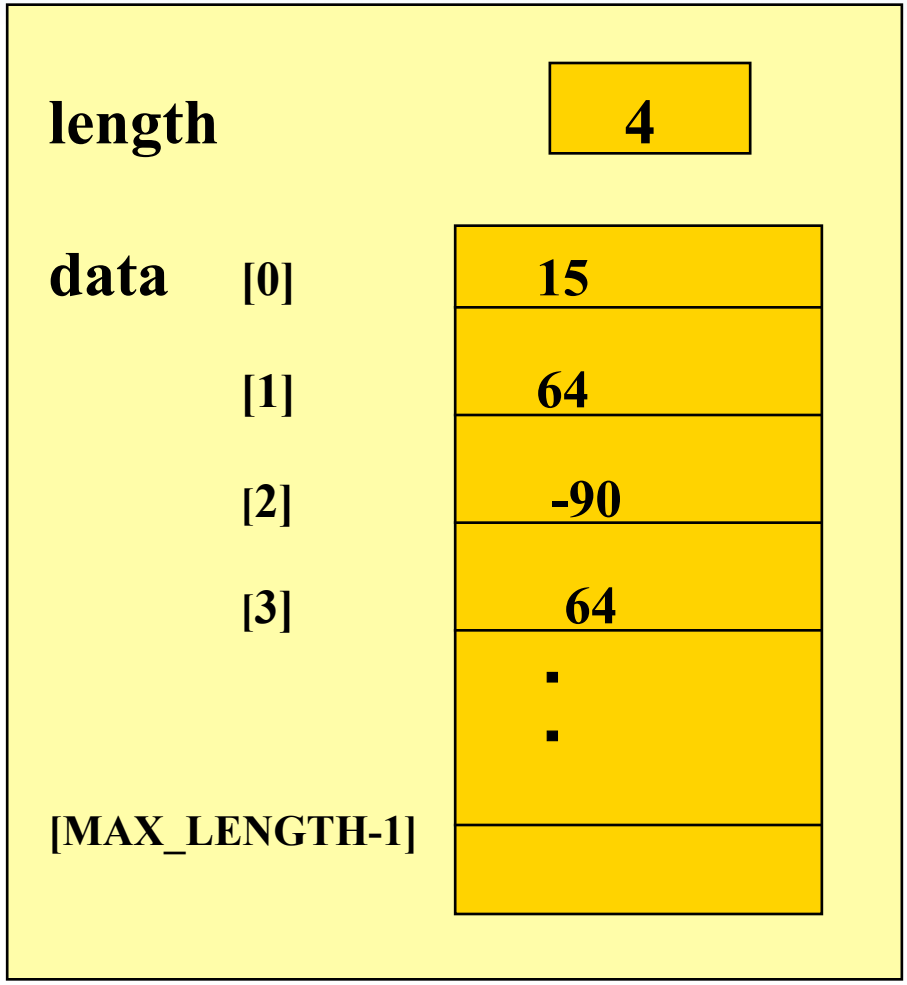
```
ItemType GetNextItem ()
// Pre: No transformer has been executed since last call
// Post:Return value is currentPos@entry
//   Current position has been updated
//   If last item returned, next call returns first item
{
    ItemType item;
    item = data[currentPos];
    if (currentPos == length - 1)
        currentPos = 0;
    else
        currentPos++;
    return item;
}
```

Iteration Operator

```
{
    ItemType item;
    item = data[currentPos];
    if (currentPos == length - 1)
        currentPos = 0;
    else
        currentPos++;
    return item;
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Reset

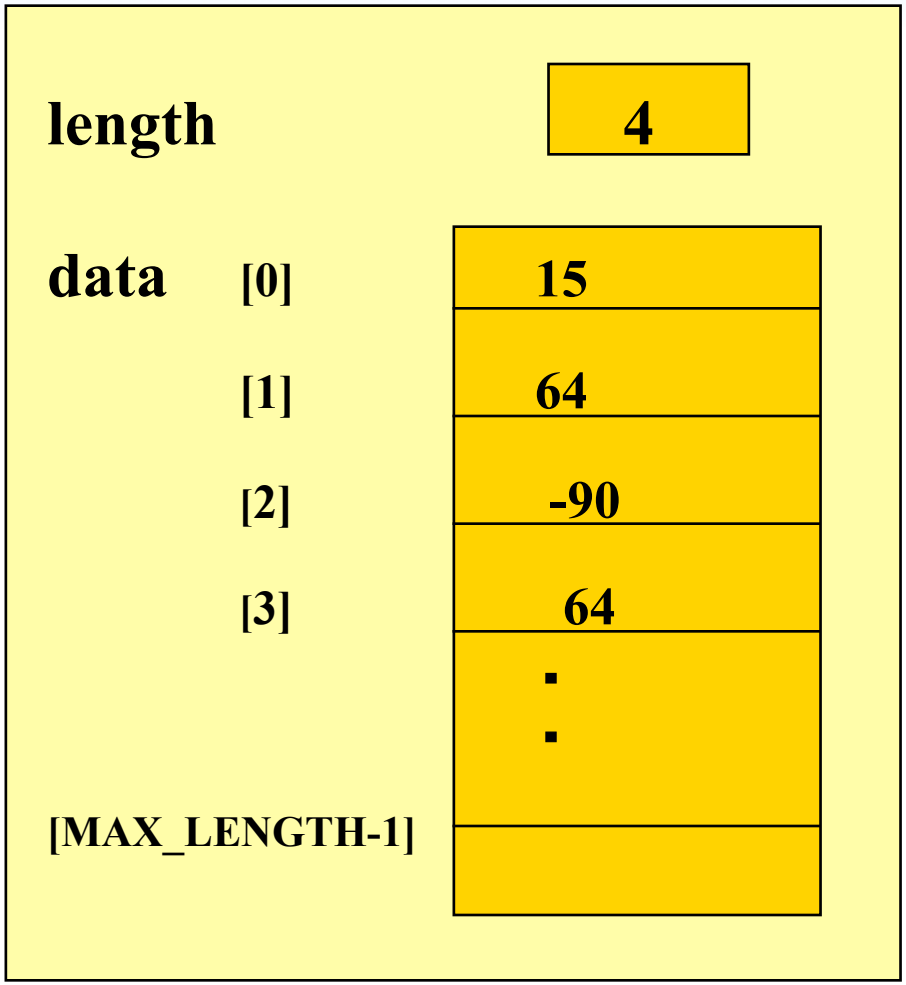


currentPos: 0

item ?

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

GetNextItem



currentPos: 1

currentPos is incremented
item is returned

item 15

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Selection Sort Process

Selection sort

- Examines the entire list to select the smallest element
- Places that element where it belongs (with array subscript 0)
- Examines the remaining list to select the smallest element from it

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Selection Sort Process, cont...

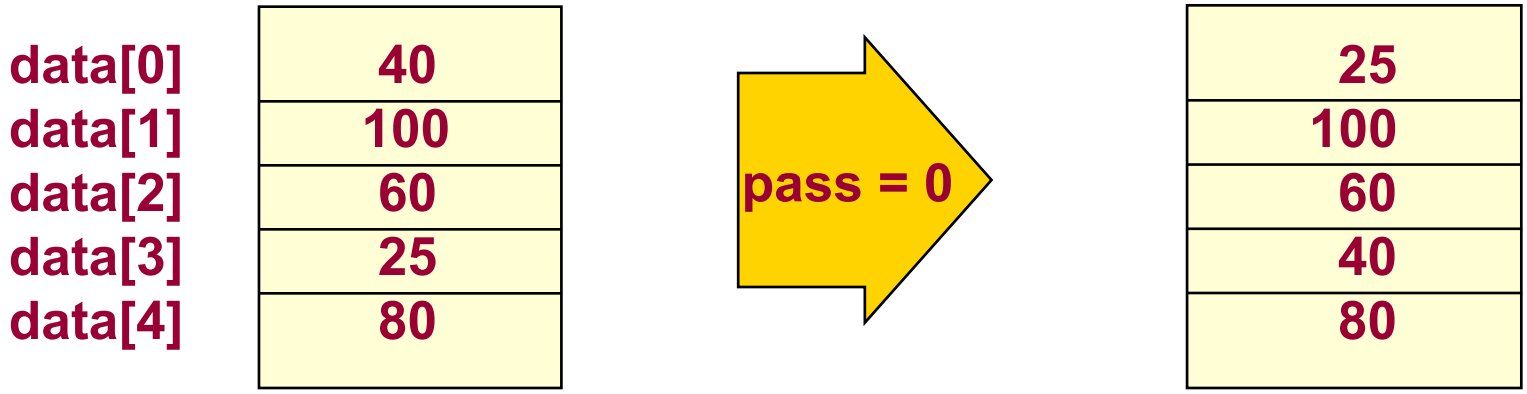
- Places that element where it belongs (with array subscript 1)
- Examines the last 2 remaining list elements to select the smallest one
- Places that element where it belongs in the array

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Selection Sort Algorithm

FOR passCount going from 0 through length - 2
Find minimum value in data[passCount . . length-1]
Swap minimum value with data[passCount]

length = 5



Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
void  List::SelSort ()
// Sorts list into ascending order
{
    ItemType temp;
    int passCount;
    int sIndx;
    int minIndx; // Index of minimum so far
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
for (passCount = 0; passCount < length - 1;
        passCount++)
    {
        minIndx = passCount;
        // Find index of smallest value left
        for (sIndx = passCount + 1;
              sIndx <  length; sIndx++)
            if (data[sIndx] = data[minIndx])
                minIndx = sIndx;
        temp = data[minIndx];       // Swap
        data[minIndx] = data[passCount];
        data[passCount] = temp;
    }
}
```

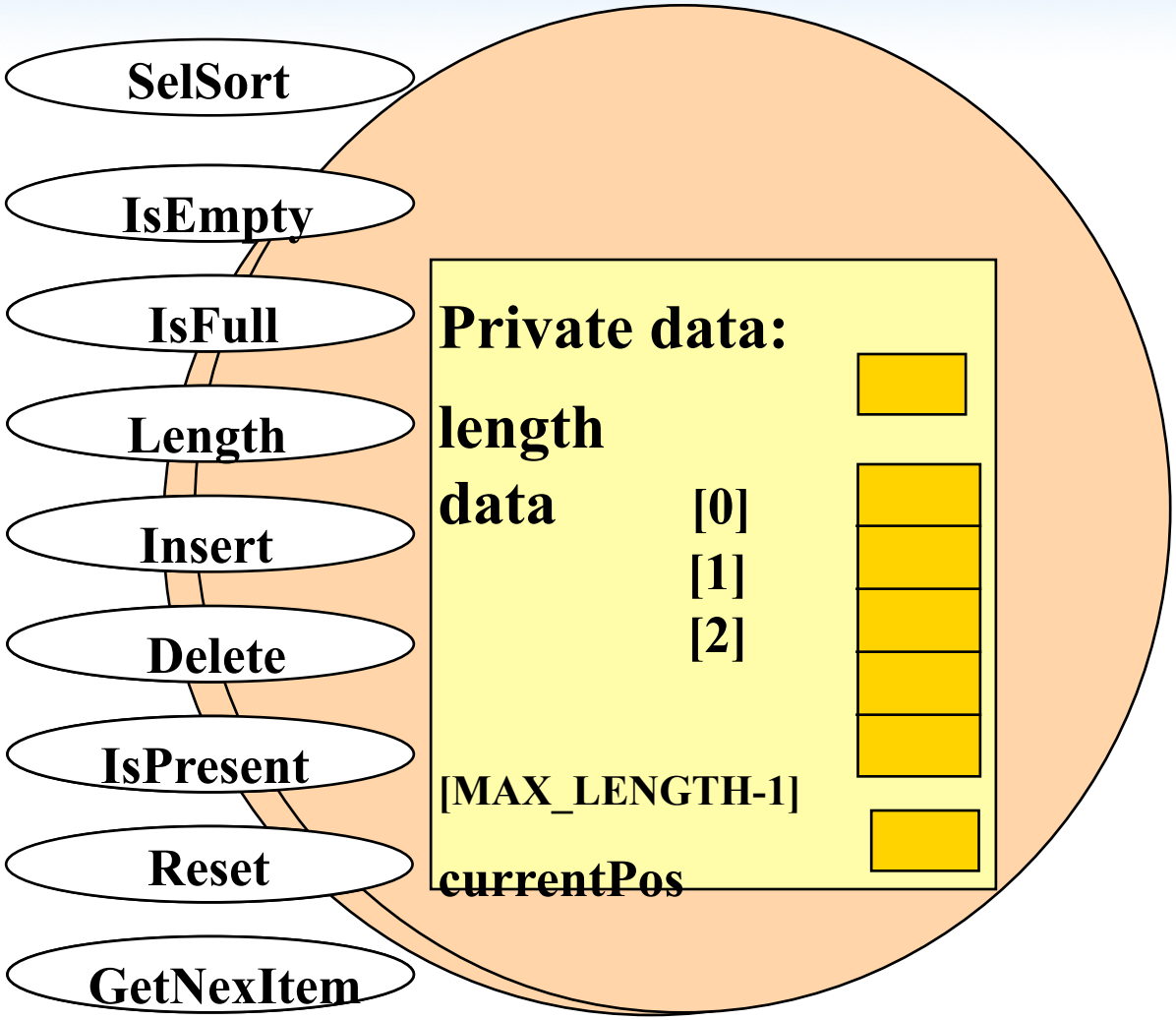
Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Recall:
Sorted and Unsorted Lists

UNSORTED LIST	SORTED LIST
Elements are placed into the list in no particular order	List elements are ordered in some way -- either numerically or alphabetically

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Array-based class SortedList



Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
// Specification file sorted list (“slist.h”)
const  int  MAX_LENGTH  =  50;
typedef int   ItemType;

class SortedList       // Declares a class data type
{
public:                // Public member functions

    List();            // constructor
    bool IsEmpty () const;
    bool IsFull ()  const;
    int  Length ()  const; // Returns length of list
    void Insert (ItemType  item);
    void Delete (ItemType  item);
    bool IsPresent(ItemType  item)  const;
    void SelSort ();
    void Reset ();
    ItemType GetNextItem ();
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
private: // Private data members
    // Number of values currently stored
    int length;
    ItemType data[MAX_LENGTH];
    int  CurrentPos;  // Used in iteration
};
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
//  SPECIFICATION FILE     ARRAY-BASED SORTED LIST
      (slist.h)
const  int  MAX_LENGTH  =  50;
typedef   int   ItemType;

class  SortedList
{
public: // public member functions
   SortedList ();  // constructor
   bool IsEmpty () const;
   bool IsFull ()  const;
   int  Length ()  const; // returns length of list
   void Insert (ItemType  item);
   void Delete (ItemType  item);
   bool IsPresent(ItemType  item)  const;
   void Print ();
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
private: //  private data members

  int length; // number of values currently stored
  ItemType  data[MAX_LENGTH];
  void   BinSearch ( ItemType item, bool& found, int&
  position) const;
};
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Member Functions

Which member function specifications and implementations must change to ensure that any instance of the SortedList ADT remains sorted at all times?

- **Insert**
- **Delete**

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Insert Algorithm for SortedList ADT

- **Create space** for the new item by shifting down all the larger list elements
- **Put** the new item in the list
- **Increment** length

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Implementing `SortedList` Member Function `Insert`

```
// Implementation file ("slist.cpp")

void SortedList::Insert (/* in */  ItemType  item)
// Pre: length < MAX_LENGTH && item is assigned
//     && data[0 . . length-1] are in
//     ascending order
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Implementing `SortedList` Member Function `Insert`

```
// Post: item is in the list && length ==
//    length@entry + 1 && data[0 . . length-1] are
//    in ascending order
{
            .
            .
            .

}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
void  SortedList::Insert (ItemType  item)
{
    int index;
    // Find proper location for new element
    index  =  length - 1;
    // Starting at bottom of array shift down
    //  values larger than item to make room for
    //  new item
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
while (index >= 0  &&  item < data[index] )
    {
        data[index + 1]  =  data[index];
        index--;
     }
    // Insert item into array
    data[index] = item;
    length++;
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Delete Algorithm for SortedList ADT

- **Find** the position of the element to be deleted from the sorted list
- **Eliminate** space occupied by the item being deleted by shifting up all the larger list elements
- **Decrement** length

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Implementing `SortedList` Member Function `Delete`

```
void  SortedList::Delete (/* in */  ItemType  item)
//  Deletes item from list, if it is there
//  Pre: 0 < length <= INT_MAX/2 && item is assigned
//      && data[0 . . length-1] are in ascending order
// Post: IF item is in data array at entry
//      First occurrence of item is no longer in array
//      && length == length@entry-1
//      && data[0 . . Length-1] are in ascending order
//        ELSE
//      length and data array are unchanged
{
              .
              .
              .
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Implementing `SortedList` Member Function `Delete`

```
// Post: IF item is in data array at entry
//       First occurrence of item is no longer
//       in array
//       && length == length@entry-1
//       && data[0 . . Length-1] are in
//       ascending order
//         ELSE
//       length and data array are unchanged
{
              .
              .
              .
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
void SortedList::Delete (/* in */  ItemType  item)
{
    bool found;    // true, if item is found
    int  position; // Position of item, if found
    int  index;
    // Find location of element to be deleted
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
BinSearch (item, found, position);
    if (found)
    {
      // Shift elements that follow in sorted list

        for (index = position; index < length + 1;
             index++)
            data[index ] = data[index  + 1];
         length--;
    }
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Improving Member Function `IsPresent`

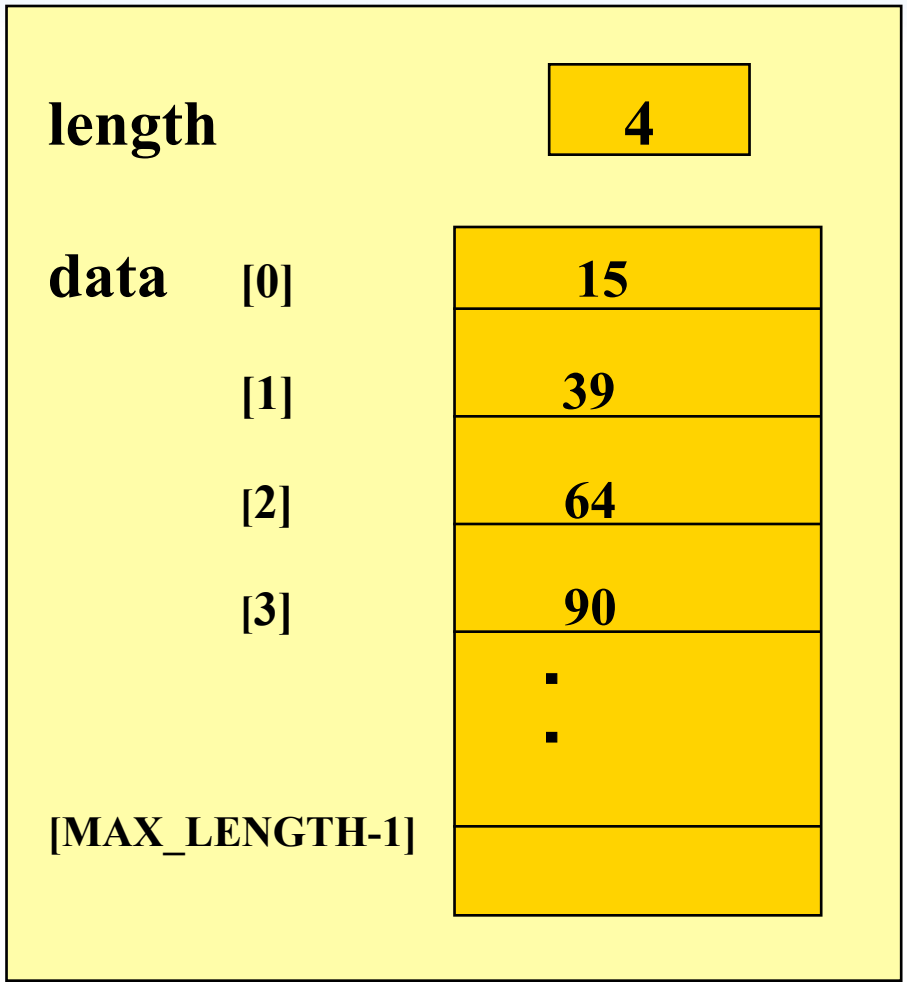
Recall that with the unsorted List ADT we examined each list element beginning with data[0], until we either found:

A match with item or we had examined all the elements in the unsorted List

How can the searching algorithm be improved for SortedList ADT?

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Searching for 55 in a SortedList



A sequential search for 55 can stop when 64 has been examined.

item 55

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Binary Search in SortedList

- Examines the element in the middle of the array
  - Is it the sought item? If so, stop searching
  - Is the middle element too small? Then start looking in second half of array
  - Is the middle element too large? Then begin looking in first half of the array

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Binary Search in SortedList

- Repeat the process in the half of the data that should be examined next
- Stop when item is found or when there is nowhere else to look

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
void  SortedList::BinSearch (ItemType  item,   bool&  found,
   int&  position)
// Searches sorted list for item, returning position of item,
//  if item was found
{
    int middle;
    int first  =  0;
    int last  = length - 1;
    found = false;
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
while (last >= first  &&   !found)
     {  middle = (first + last)/2; // Index of middle element

        if (item  <  data[middle])
            last = middle - 1;     // Look in first half next
        else if (item  >  data[middle])
           first = middle + 1;     // Look in second half next
        else
           found = true;           // Item  has been found
     }
     if  (found)
       position = middle;
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Trace of Binary Search

item = 84

15	26	38	57	62	78	84	91	108	119
data[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
first				middle					last

item > data[middle] → first = middle + 1

~~15~~	~~26~~	~~38~~	~~57~~	~~62~~	78	84	91	108	119
data[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
					first		middle		last

item < data[middle] → last = middle - 1

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Trace continued

item = 84

~~15~~	~~26~~	~~38~~	~~57~~	~~62~~	78	84	~~91~~	~~108~~	~~119~~
data[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

first, middle

last

item > data[middle] → first = middle + 1

~~15~~	~~26~~	~~38~~	~~57~~	~~62~~	~~78~~	84	~~91~~	~~108~~	~~119~~
data[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

first, last, middle

item == data[middle] → found = true

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Another Binary Search Trace

item = 45

15	26	38	57	62	78	84	91	108	119
data[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
first				middle					last

item < data[middle] → last = middle - 1

15	26	38	57	~~62~~	~~78~~	~~84~~	~~91~~	~~108~~	~~119~~
data[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
first	middle		last						

item > data[middle] → first = middle + 1

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Trace continued

item = 45

~~15~~	~~26~~	38	57	~~62~~	~~78~~	~~84~~	~~91~~	~~108~~	~~119~~
data[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

first, middle

last

item > data[middle] → first = middle + 1

~~15~~	~~26~~	~~38~~	57	~~62~~	~~78~~	~~84~~	~~91~~	~~108~~	~~119~~
data[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

first, middle, last

item < data[middle] → last = middle - 1

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

item = 45

Trace concludes

~~15~~	~~26~~	~~38~~	~~57~~	62	78	~~84~~	~~91~~	108	119
data[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
		last	first						

first > last ⇒ found = false

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Still More Efficient `IsPresent`

```
bool  SortedList::IsPresent
  (/* in */  ItemType  item)  const
// Searches list for item, reporting whether found
// Pre: length <= INT_MAX/2 && item is assigned
//    &&  data[0 . . length-1] are in ascending order
// Post: Return value == true, if item is in
//   data[0 . . length-1] == false, otherwise
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Still More Efficient `IsPresent`

```
{
    bool found;
    int  position;

    BinSearch (item, found, position);

    return  found;
}
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Comparison of Sequential and Binary Searches

	Average Number of Iterations to Find `item`	
Length	***Sequential Search***	***Binary Search***
10	**5.5**	**2.9**
100	**50.5**	**5.8**
1,000	**500.5**	**9.0**
10,000	**5000.5**	**12.4**

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Order of Magnitude of a Function

The order of magnitude, or Big-O notation, of an expression describes the complexity of an algorithm according to the highest order of N that appears in its complexity expression

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Names of Orders of Magnitude

$O(1)$	constant time
$O(\log_2 N)$	logarithmic time
$O(N)$	linear time
$O(N^2)$	quadratic time
$O(N^3)$	cubic time

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

N	$\log_2 N$	$N*\log_2 N$	N^2
1	0	0	1
2	1	2	4
4	2	8	16
8	3	24	64
16	4	64	256
32	5	160	1024
64	6	384	4096
128	7	896	16,384

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Big-O Comparison of List Operations

OPERATION	UnsortedList	SortedList
IsPresent	O(N)	O(N) sequential search $O(\log_2 N)$ binary search
Insert	O(1)	O(N)
Delete	O(N)	O(N)
SelSort	$O(N^2)$	

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

In Addition . . .

To the string class from the standard library accessed by #include <string>

C++ also has another library of string functions for C strings that can be accessed by #include <cstring>

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

What is a C String?

A C string is a char array terminated by the null character '\0' (with ASCII value 0)

A C string variable can be initialized in its declaration in two equivalent ways.

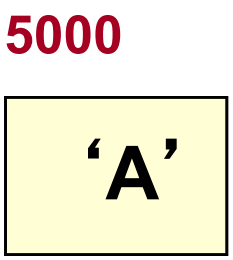
```
char message[8] = { 'H', 'e', 'l', 'l', 'o', '\0' };

char  message[8]  =  "Hello";
```

'H'	'e'	'l'	'l'	'o'	'\0'		
message[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

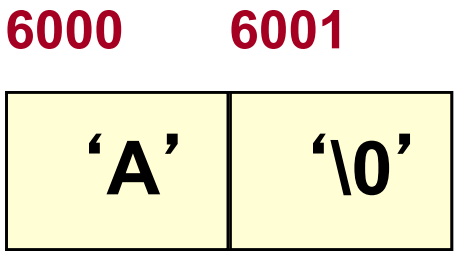
Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

`char` vs. C string

'A' has data type `char`
and is stored in 1 byte



"A" is a C string of 2 characters
and is stored in 2 bytes

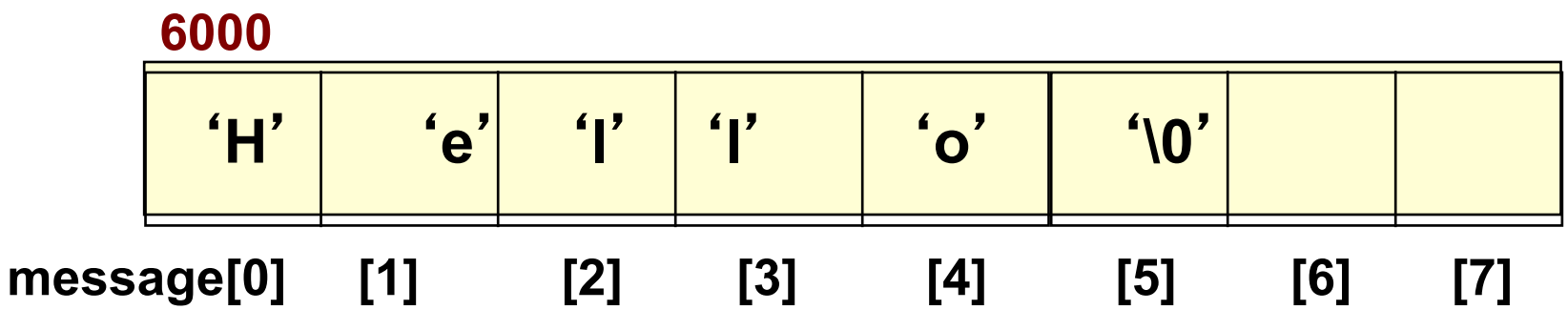


Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Recall that . . .

```
char message[8];
 // Declaration allocates memory
```

To the compiler, the value of the identifier message is the base address of the array. We say `message` is a pointer (because its value is an address). It "points" to a memory location.



Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Aggregate C String I/O in C++

I/O of an entire C string is possible using the array identifier with no subscripts and no looping.

EXAMPLE

```
char   message[8];
cin  >>  message;
cout  <<  message;
```

However . . .

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Extraction operator >>

When using the extraction operator (>>) to read input characters into a string variable, the following things happen:

- The >> operator skips any leading whitespace characters such as blanks and newlines
- It then reads successive characters into the array

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Extraction Operator >>

- And the >> operator stops at the first trailing whitespace character (which is not consumed, but remains waiting in the input stream)
- The >> operator adds the null character to the end of the string

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Example Using >>

```
char  name[5];
cin  >>  name;
```

total number of elements in the array

Suppose input stream looks like this:

□ □ J o e □

7000

'J'	'o'	'e'	'\0'	
name[0]	name[1]	name[2]	name[3]	name[4]

null character is added

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Function `get()`

- Because the extraction operator stops reading at the first trailing whitespace, >> cannot be used to input a string with blanks in it
- If your string's declared size is not large enough to hold the input characters and add the '\0', the extraction operator stores characters into memory beyond the end of the array
- Use `get` function with two parameters to overcome these obstacles

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Example of Function `get()`

```
char  message[8];
cin.get (message, 8);
// Inputs at most 7 characters plus '\0'
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

inFileStream.get (str, count + 1)

- `get` does not skip leading whitespace characters such as blanks and newlines

- `get` reads successive characters (including blanks) into the array

- get stops when it either has read count characters, or it reaches the newline character '\n', whichever comes first

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

inFileStream.get (str, count + 1)

- `get` appends the null character to str
- If newline is reached, it is not consumed by `get`, but remains waiting in the input stream

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Function `ignore()`

- `ignore` can be used to consume any remaining characters up to and including the newline '\n' left in the input stream by `get`

```
cin.get(string1, 81);
  // Inputs at most 80 characters
  cin.ignore(30, '\n');
  // Skips at most 30 characters
  //  but stops if '\n' is read
  cin.get(string2, 81);
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

Another Example Using `get()`

```
char  ch;
char  fullName[31];
char  address[31];
cout << "Enter your full name: ";
cin.get (fullName, 31);
cin.get (ch);  // To consume the newline
cout << "Enter your address: ";
cin.get (address, 31);
```

'N'	'e'	'l'	'l'	' '	'D'	'a'	'l'	'e'	'\0'	. . .

fullName[0]

'A'	'u'	's'	't'	'i'	'n'	' '	'T'	'X'	'\0'	. . .

address[0]

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

String Function Prototypes in <cstring >

```
int   strlen (char  str[]);
//  FCTNVAL     ==  integer length of string str (not including '\0')
```

```
int   strcmp (char  str1[], char  str2[]);
//  FCTNVAL     ==  negative, if str1 precedes str2 lexicographically
//                  ==  positive, if str1 follows str2 lexicographically
//                  ==  0, if str1 and str2 characters same through '\0'
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

String Function Prototypes in <cstring >, cont...

```
char * strcpy (char  toStr[],  char  fromStr[]);
//  FCTNVAL     ==  base address of toStr (usually ignored)
// POSTCONDITION:     characters in string fromStr are copied to
//                    string toStr, up to and including '\0',
//                    overwriting contents of string toStr
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
# include  <cstring >
  .
  .
  .

 char author[21];
 int  length;

 cin.get (author, 21);
 length = strlen (author);
 // What is the value of length ?
```

5000

‘C’	‘h’	‘i’	‘p’	‘ ’	‘W’	‘e’	‘e’	‘m’	‘s’	‘\0’	. .	. .

author[0]

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
char  myName[21] = "Huang";    // What is output?
char  yourName[21];

cout << "Enter your last name: ";
cin.get (yourName, 21);

if  (strcmp (myName, yourName) == 0)
    cout  <<  "We have the same name! ";
else if  (strcmp (myName, yourName) < 0)
    cout  <<  myName   << " comes before "
          <<  yourName;
else if  (strcmp (myName, yourName) > 0)
    cout  <<  yourName  << "comes before "
          <<  myName;
```

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

myName[0]

‘H’	‘u’	‘a’	‘n’	‘g’	‘\0’					. . .

yourName[0]

‘H’	‘e’	‘a’	‘d’	‘i ‘	‘n’	‘ g’	‘t’	‘o’	‘n’	‘\0’	. . .

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
char  myName[21] = "Huang";
char  yourName[21];

if (myName  ==  yourName)
// Compares addresses only!
//   That is, 4000 and 6000 here.
//   == does not compare contents!
{
    .
}
```

4000 myName[0]

'H'	'u'	'a'	'n'	'g'						. . .

6000 yourName[0]

'H'	'e'	'a'	'd'	'i'	'n'	'g'	't'	'o'	'n'	'\0'	. . .

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
char  myName[21] = "Huang";
char  yourName[21];
```

```
cin.get (yourName, 21);
yourName  =  myName;
```

What happens?

4000

'H'	'u'	'a'	'n'	'g'	'\0'					. . .

myName[0]

6000

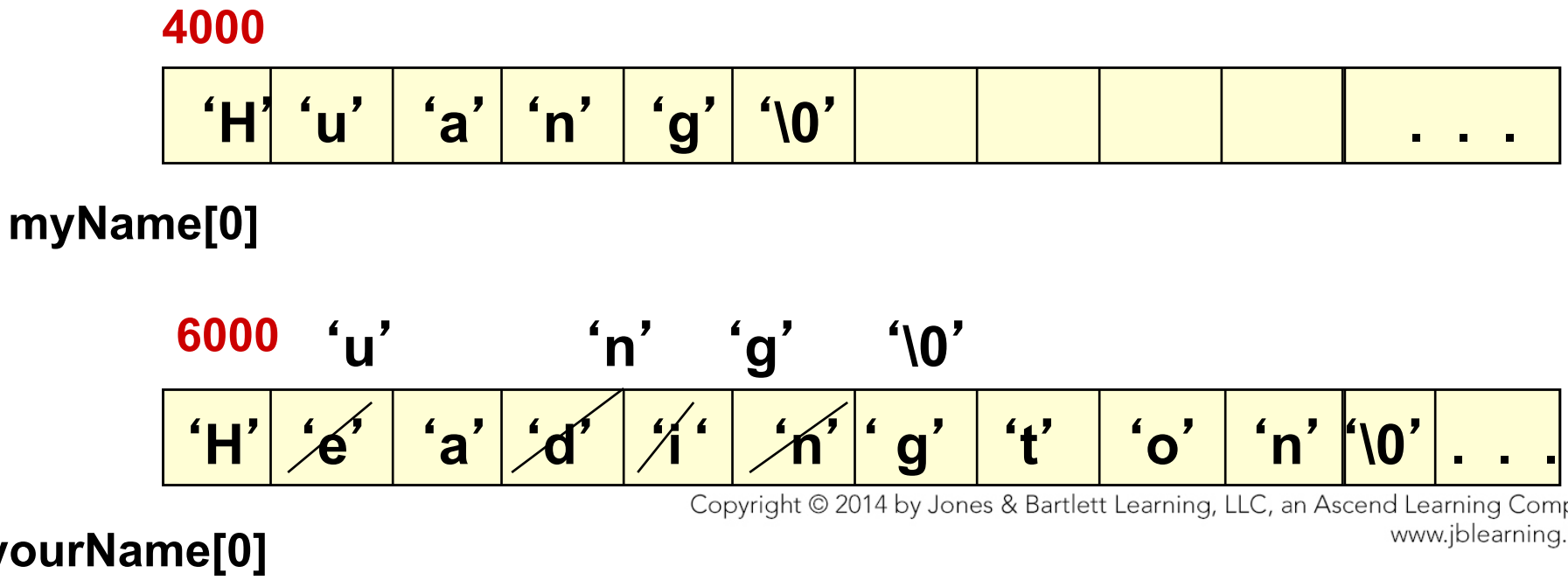
'H'	'e'	'a'	'd'	'i'	'n'	'g'	't'	'o'	'n'	'\0'	. . .

yourName[0]

Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

```
char  myName[21]  = "Huang";
char  yourName[21];

cin.get (yourName, 21);
strcpy (yourName, myName);
```

What happens?

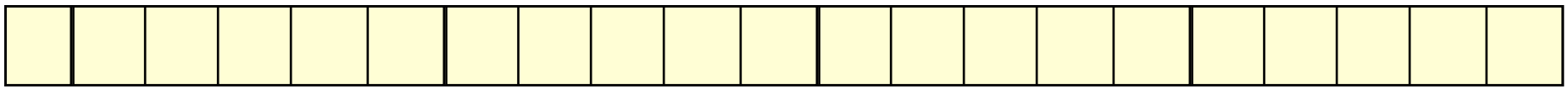


Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com

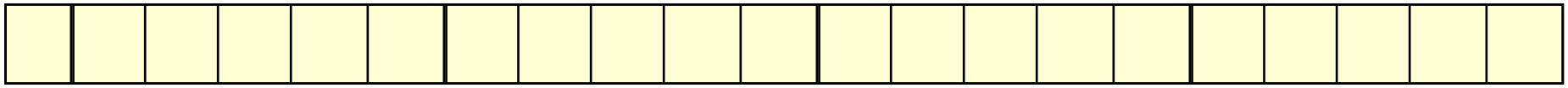
Using `typedef` with Arrays

```
typedef char String20[21];
// Names String20 as an array type

String20   myName;   // These declarations allocate
String20   yourName; //  memory for three  variables
bool isSeniorCitizen;
```

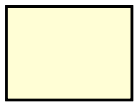
5000



6000



7000



Copyright © 2014 by Jones & Bartlett Learning, LLC, an Ascend Learning Company
www.jblearning.com