

CSE1301 Computer Programming Lecture 33: List Manipulation

Recall

- List
 - an array of the same type of elements
- Linear Search
 - Advantage: list can be unsorted
 - Disadvantage: $O(N)$
- Binary Search
 - Advantage: $O(\log_2 N)$
 - Disadvantage: list must be sorted

20

Topics

- How can we maintain a list?
 - unsorted
 - sorted
- Operations on a list:
 - initialize the list
 - add an element
 - delete an element
 - find an element
 - print the list
- Add, Find and Delete: *sorted or unsorted* ?

31

List: Example

```
#define MAX_COUNT 6 /* Max number of items */
#define MAX_LEN 20 /* Max length of a name */

struct nameListRec
{
    int count; /*Number of items currently in the list*/
    char names[MAX_COUNT][MAX_LEN];
};
typedef struct nameListRec nameList;
nameList students;
```

students.count: ???

students.names

???	???	???	???	???	???
0	1	2	3	4	5

41

List: Initialize

```
nameList newList ( )
{
    nameList students;
    students.count = 0;
    return students;
}
```

students.count: 0

students.names

???	???	???	???	???	???
0	1	2	3	4	5

51

Unsorted List: Add

```
strcpy(students.names[ students.count ], "Dave");
students.count++;
```

students.count: 0

students.names

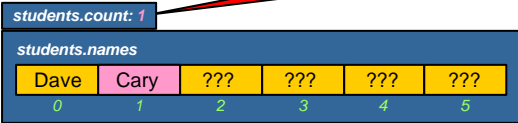
Dave	???	???	???	???	???
0	1	2	3	4	5

61

Unsorted List: Add (cont)

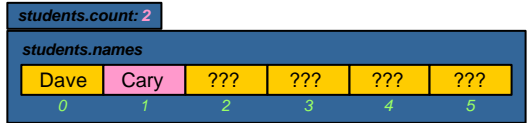
```
strcpy(students.names[ students.count ], "Dave");
students.count++;
strcpy(students.names[ students.count ], "Cary");
```

next available position



Unsorted List: Add (cont)

```
strcpy(students.names[ students.count ], "Dave");
students.count++;
strcpy(students.names[ students.count ], "Cary");
students.count++;
```



Unsorted List: Add (cont)

```
nameList addUnsorted ( nameList students, char *newname )
{
  strcpy(students.names[students.count], newname);
  students.count++;
  return students;
}
```

```
...
nameList csel301;
...
csel301 = newList();
csel301 = addUnsorted(csel301, "Dave");
csel301 = addUnsorted(csel301, "Cary");
...
```

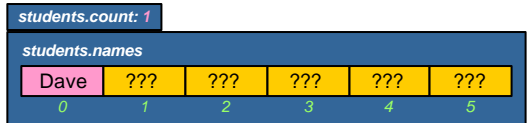
9

Sorted List: Add

- Case 1: List is empty
 - same as unsorted

```
strcpy(students.names[students.count], "Dave");
students.count++;
```

Example: add "Dave"



10

Sorted List: Add (cont)

- Case 2: List is not empty
 - find correct position

Example: add "Abe"

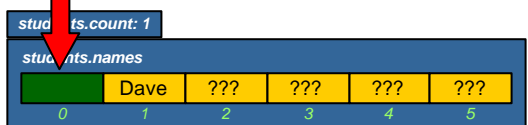


11

Sorted List: Add (cont)

- Case 2: List is not empty
 - find correct position
 - make room by moving all items to the right

Example: add "Abe"

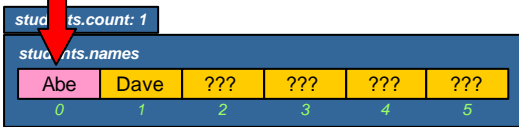


12

Sorted List: Add (cont)

- Case 2: List is not empty
 - find correct position
 - make room by moving all items to the right
 - put item in that position

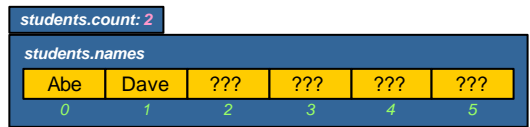
Example: add "Abe"



Sorted List: Add (cont)

- Case 2: List is not empty
 - find correct position
 - make room by moving all items to the right
 - put item in that position
 - update count

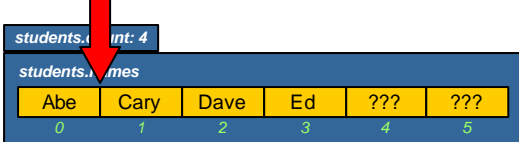
Example: add "Abe"



Sorted List: Add (cont)

- Case 2: List is not empty

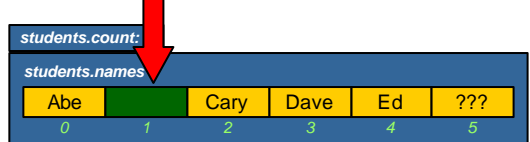
Example: add "Arnie"



Sorted List: Add (cont)

- Case 2: List is not empty

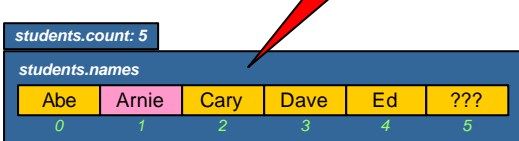
Example: add "Arnie"



Sorted List: Add (cont)

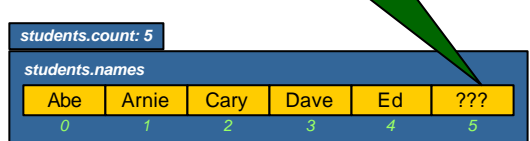
- Case 2: List is not empty

Example: add "Arnie"



Sorted List: Add (cont)

- Case 2: List is not empty



Algorithm addElement

```

count addElement (array, last, val)
{
  if list is full
  {
    print message and exit
  }
  else if list is empty
  {
    add val in first position of array
    increment last
  }
  else
  {
    while ( end of array not reached )
    {
      if ( val < current element )
      {
        shuffle to the right the array
        elements starting at current element
        break loop
      }
    }
    insert val in the empty spot of array
    update last
  }
  return position of added element
}

```

1919

Function addElement

```

int addElement( int *arr,
int *last, int val,
int size )
{
  int count, i;
  if (*last == size-1)
  { /* list is full */
    printf("List is full\n");
    return -1;
  }
  else if (*last == -1)
  { /* list is empty */
    arr[0] = val;
    (*last)++;
    return 0;
  }
  else
  {
    for(count=0; count <= *last;
count++)
    { /* add before first larger
value */
      if ( val < arr[count] )
      { /* shuffle along starting
at the end */
        for ( i = *last;
i >= count; i--)
        {
          arr[i+1]=arr[i];
        }
        break;
      }
    }
    arr[count] = val;
    (*last)++;
    return count;
  }
}

```

2010

addElement -- Exercise

- Modify the C program so that it works on the struct nameList
 - Version 1:
 - parameters: new item, count and array of strings
 - return value: whether the list is full
 - Version 2:
 - parameters: new item and a nameList structure
 - return value: updated struct
 - use a function called isListFull to determine if the list is full

2121

List: Add an Element

- Unsorted list
 - append after the last element
- Sorted list
 - find correct position
 - make room for the new element
- Special cases:
 - list is empty
 - list is full
 - item to be inserted is already in the list

2212

List: Print (Version 1)

```

void printList ( int count, char nomen[][MAX_LEN] )
{
  int i;

  for (i=0; i < count; i++)
  {
    printf("%d %s\n", i, nomen[i]);
  }
}

```



Array of strings

```
printList ( students.count, students.names );
```

2323

List: Print (Version 2)

```

void printList ( nameList class )
{
  int i;

  for (i=0; i < class.count; i++)
  {
    printf("%d %s\n", i, class.names[i] );
  }
}

```

```
printList ( students );
```

2414

List: Find an Element

- Sorted
 - Use binary search
- Unsorted
 - Use linear search
- *Used for deleting an element*

2525

List: Delete an Element

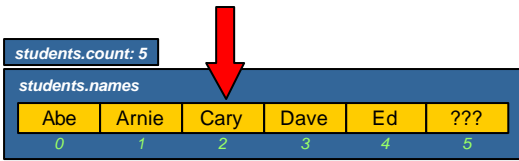
- Same for both sorted and unsorted lists
- Algorithm:
 - find the position of the item to be deleted
 - linear or binary search
 - if found
 - move all items one position to the left
 - decrement count
- Special cases:
 - list is empty
 - item not found

2626

List: Delete an Element

- find the position of the element

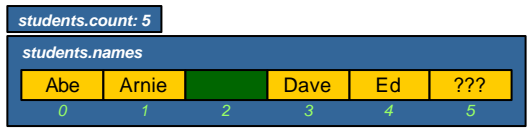
Example: delete "Cary"



List: Delete an Element

- find the position of the element
- remove the element

Example: delete "Cary"

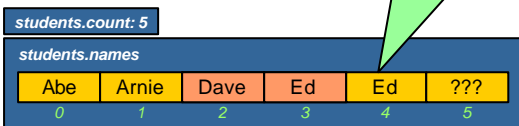


List: Delete an Element

- find the position of the element
- remove the element
- shuffle the items after the deleted item to the left

Example: delete "Cary"

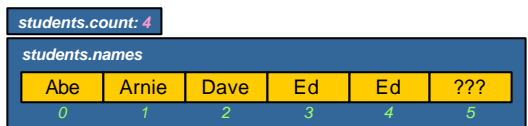
We don't need to actually clear this. Just update count



List: Delete an Element

- find the position of the element
- remove the element
- shuffle the items after the deleted item to the left
- decrement count

Example: delete "Cary"



Summary

- Operations on lists
 - initialize, add element, delete element, find element, print
- Add, Find and Delete: sorted or unsorted?
- Review: array of strings and structs as parameters

3131