

Chapter 13

Lists

List

- List
 - A variable-length, linear collection of homogeneous components
- Example
 - `StudentRec Students[100];`
 - A list of 100 students structures

Common Operations

- On almost all data structures, there are 4 common activities
 1. Insertion
 2. Deletion
 3. Searching
 4. Sorting
 - Sorting can be handled at insertion or deferred until a later time.

Insertion

- Insertion into an unordered list simply requires placing the next record at the end of the list.
- Insertion into an order list requires finding its location and making a space to place the item.

Deletion

- You must find the record you desire and remove it from the list. In an array based list, this will require some shifting.
- If order is unimportant, you can simply swap it with the last record and reduce the number of records by one
- If order is important, you must shift all the records after it down.

Searching

- A common activity on data structures is searching.
- Searching an array based list is simple

```
for (index=0; index < length && !found; index++)  
    if ( item == data[index] )  
        found = true;  
//index now contains the location of the item for  
which you were searching
```

Searching an Ordered List

- If the list is ordered, there is a better way to search.
- A Binary Search can be performed.
- This will greatly speed up your search.
- The idea is you look at the middle item.
- If it is bigger than what you are looking for, the item lies to the left of the middle item.
- If it is smaller than what you are looking for, the item lies to the right of the middle item.
- Choose the appropriate side, and go again.

Sorting

- Sorting can be done at the time of insertion, an insertion sort.
- Or after insertion.
- There are reasons to do both.
- For our purposes, we will sort after.
- Keeping an array in sorted order is not fun or worth the effort.

Selection Sort

```
for (passCount = 0; passCount<length-1;passCount++ )
{
    minIndx = passCount;
    for
    (srchIndx=passCount+1;srchIndx<length;srchIndx++)
        if ( data[srchIndx] < data[minIndx] )
            minIndx = srchIndx;
    temp = data[minIndx];
    data[minIndx] = data[passCount];
    data[passCount] = temp;
}
}
```

Bubble Sort

- There is another sort known as bubble sort.
- It is an awful sort.
- It is essentially the same sort as selection sort, except that it makes the swap anytime that it finds an element is in the wrong spot.

Binary Search

```
int first = 0;
int last = length - 1;
int middle;
found = false;
while ( last >= first && !found )
{
    middle = ( first + last ) / 2;
    if ( item < data[middle] )
        last = middle - 1;
```

```
    else if ( item > data[middle] )
        first = middle + 1;
    else if ( item == data[middle] )
        found = true;
}
if ( found )
    position = middle;
```

How does this apply to me?

- For your last project you will create a list of data items.
- You will use a struct to hold the data and an array for the list.
- The list will be unordered, but it will support sorting through the use of selection sort.
- That will require modifying what has been show slightly to work with the struct.