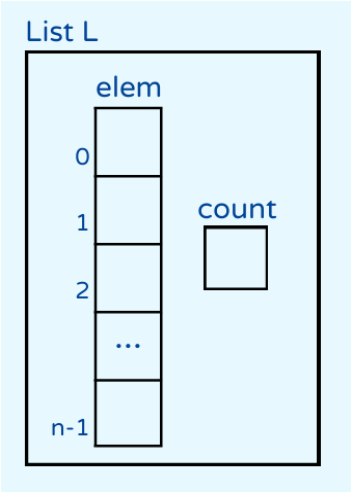
Array List

Array List is defined as an array inside a List structure that contains a respective count variable.

Variation 1

List is a **static array** and accessed by **value.**




```
#define MAX 10
typedef struct{
    int elem[MAX];
    int count;
} List;

List L;
```

Operations	Checklist	Example
`List initialize(List L);`	☐ Set the count to 0 ☐ Return List	

`List insertPos(List L, int data, int position);`	☐ **Position must be valid (less than or equal to count)**
	☐ **The array must not be full (count should not be equal to max)**
	☐ Insert the element into the specified position
	☐ Shift elements right to make space for
		`Before:`
		`  elem: [ 1, 3, 2, 5, ... ]`
		`  count: 4`
		
		`L = insertPos(L, 4, 2);`
		
		`After:`

	the position if necessary ☐ Increment count ☐ Return modified List	`elem: [ 1, 3, 4, 2, 5, ... ]` `count: 5`

`List deletePos(List L, int position);`	☐ **Position must be valid (less than or equal to count)**
	☐ Shift elements to the left to fill the position if necessary
	☐ Decrement count
	☐ Return modified List
		`Before:`
		`elem: [ 1, 3, 2, 5, ... ]`
		`count: 4`
		
		`L = deletePos(L, 1);`
		
		`After:`
		`elem: [ 1, 2, 5, ... ]`
		`count: 3`

`int locate(List L, int data);`	☐ Loop through the array and return the position of the data if found ☐ If data is not found, return -1	

`List insertSorted(List L, int data);`	☐ **Assume array is sorted**
	☐ **The array must not be full (count should not be equal to length)**
	☐ Insert the element into the correct position based on the value
	☐ Return modified List
		`Before:`
		`elem: [ 1, 3, 5, 10, ... ]`
		`count: 4`
		
		`L = insertSorted(L, 8);`
		
		`After:`
		`elem: [ 1, 3, 5, 8, 10, ... ]`
		`count: 5`

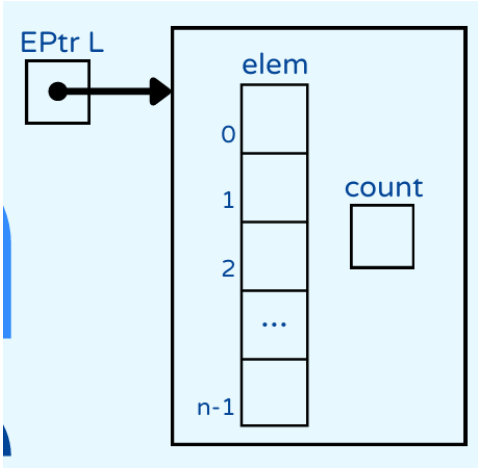
`void display(List L);`	☐ Display each element of the array until count is reached	

Note:

- For **INSERT FIRST** and **INSERT LAST**, you may use `insertPos`
- For **DELETE FIRST** and **DELETE LAST**, you may use `deletePos`
- For **DELETE BY VALUE**, you may use combination of `locate` and `deletePos`

Variation 2

List is a **static array** and accessed by **pointer.**




```
#define MAX 10
typedef struct{
    int elem[MAX];
    int count;
} Etype, *EPtr;

EPtr L;
```

Operations	Checklist	Example
`void initialize(EPtr L);`	☐ Set the count to 0	

`void insertPos(EPtr L, int data, int position);`	☐ **Position must be valid (less than or equal to count)**
	☐ **The array must not be full (count should not be equal to max)**
	☐ Insert the element into the specified position
	☐ Shift elements right to make space for the position if necessary
	☐ Increment count
		`Before:`
		`  elem: [ 1, 3, 2, 5, ... ]`
		`  count: 4`
		
		`insertPos(L, 4, 2);`
		
		`After:`
		`  elem: [ 1, 3, 4, 2, 5, ... ]`
		`  count: 5`

`void deletePos(EPtr L, int position);`	☐ **Position must be valid (less than or equal to count)**
	☐ Shift elements to the left to fill the position if necessary
	☐ Decrement count
		`Before:`
		`  elem: [ 1, 3, 2, 5, ... ]`
		`  count: 4`
		
		`deletePos(L, 1);`

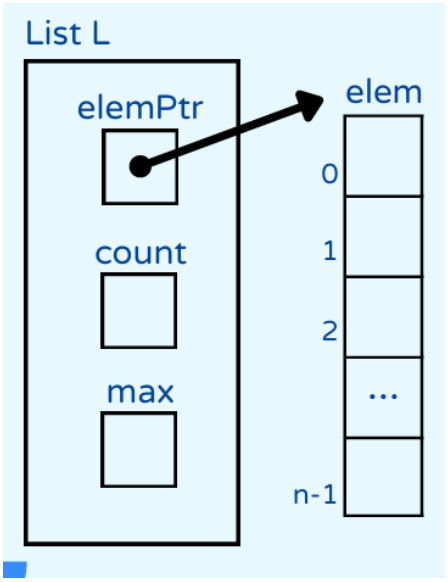
		After: elem: [1, 2, 5, ...] count: 3
`int locate(EPtr L, int data);`	☐ Loop through the array and return the position of the data if found ☐ If data is not found, return -1	
`int retrieve(EPtr L, int position);`	☐ **Position must be valid (less than or equal to count)** ☐ Return the element that is located in the given position	
`void insertSorted(EPtr L, int data);`	☐ **Assume array is sorted** ☐ **The array must not be full (count should not be equal to length)** ☐ Insert the element into the correct position based on the value	Before: elem: [1, 3, 5, 10, ...] count: 4 insertSorted(L, 8); After: elem: [1, 3, 5, 8, 10, ...] count: 5
`void display(EPtr L);`	☐ Display each element of the array until count is reached	
`void makeNULL(EPtr L);`	☐ Free the memory allocated to the List	

Note:

- For **INSERT FIRST** and **INSERT LAST**, you may use `insertPos`
- For **DELETE FIRST** and **DELETE LAST**, you may use `deletePos`
- For **DELETE BY VALUE**, you may use combination of `locate` and `deletePos`

Variation 3

List is a **dynamic array** and accessed by **value.**




```
#define LENGTH 10
typedef struct{
    int  *elemPtr;
    int count;
    int max;
} List;
```

Operations	Checklist	Example
`List initialize(List L);`	☐ Dynamically allocate memory for the array using LENGTH ☐ Set max to defined LENGTH ☐ Set the count to 0 ☐ Return List	

`List insertPos(List L, int data, int position);`	☐ **Position must be valid (less than or equal to count)**
	☐ **If the array is full, call the resize function**
	☐ Insert the element into the specified position
	☐ Shift elements right to make space for the position if necessary
	☐ Increment count
	☐ Return modified List
		`Before:`
		`  elem: [ 1, 3, 2, 5 ]`
		`  count: 4`
		`  max: 4`
		
		`L = insertPos(L, 4, 2);`
		
		`After:`
		`  elem: [ 1, 3, 4, 2, 5, ... ]`
		`  count: 5`

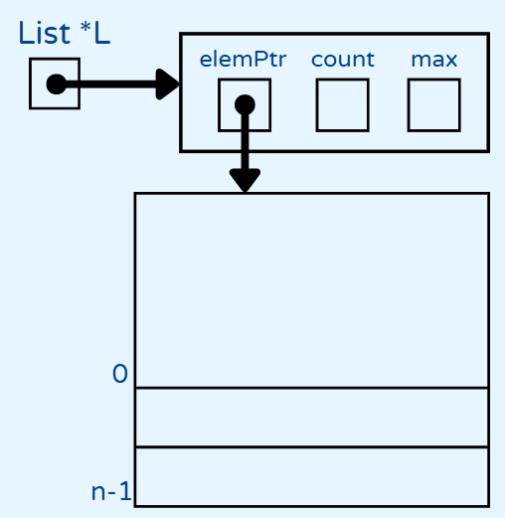
		max: 8
`List deletePos(List L, int position);`	☐ **Position must be valid (less than or equal to count)** ☐ Shift elements to the left to fill the position if necessary ☐ Decrement count ☐ Return modified List	Before: elem: [1, 3, 2, 5] count: 4 max: 4 L = deletePos(L, 1); After: elem: [1, 2, 5, ...] count: 3 max: 4
`int locate(List L, int data);`	☐ Loop through the array and return the position of the data if found ☐ If data is not found, return -1	
`List insertSorted(List L, int data);`	☐ **Assume array is sorted** ☐ **If the array is full, call the resize function** ☐ Insert the element into the correct position based on the value ☐ Return modified List	Before: elem: [1, 3, 5, 10] count: 4 max: 4 L = insertSorted(L, 8); After: elem: [1, 3, 5, 8, 10, ...] count: 5 max: 8
`void display(List L);`	☐ Display each element of the array until count is reached	
`List resize(List L);`	☐ **Reallocate the memory assigned to the array and double its length** ☐ Double the max variable ☐ Return modified List	

Note:

- For **INSERT FIRST** and **INSERT LAST**, you may use `insertPos`
- For **DELETE FIRST** and **DELETE LAST**, you may use `deletePos`
- For **DELETE BY VALUE**, you may use combination of `locate` and `deletePos`

Variation 4

List is a **dynamic array** and accessed by **pointer.**




```
#define LENGTH 10
typedef struct{
    int *elemPtr;
    int count;
    int max;
} List;
```

Operations	Checklist	Example
`void initialize(List *L);`	☐ Dynamically allocate memory for the array using LENGTH ☐ Set max to defined LENGTH ☐ Set the count to 0	

`void insertPos(List *L, int data, int position);`	☐ **Position must be valid (less than or equal to count)**
	☐ **If the array is full, call the resize**
		`Before:`
		`  elem: [ 1, 3, 2, 5 ]`
		`  count: 4`

	function
	☐ Insert the element into the specified position
	☐ Shift elements right to make space for the position if necessary
	☐ Increment count
		`max: 4`
		
		`insertPos(&L, 4, 2);`
		
		`After:`
		`elem: [ 1, 3, 4, 2, 5, ... ]`
		`count: 5`
		`max: 8`

`void deletePos(List *L, int position);`	☐ **Position must be valid (less than or equal to count)**
	☐ Shift elements to the left to fill the position if necessary
	☐ Decrement count
		`Before:`
		`elem: [ 1, 3, 2, 5 ]`
		`count: 4`
		`max: 4`
		
		`deletePos(&L, 1);`
		
		`After:`
		`elem: [ 1, 2, 5, ... ]`
		`count: 3`
		`max: 4`

`int locate(List *L, int data);`	☐ Loop through the array and return the position of the data if found ☐ If data is not found, return -1	
`int retrieve(List *L, int position);`	☐ **Position must be valid (less than or equal to count)** ☐ Return the element that is located in the given position	

`void insertSorted(List *L, int data);`	☐ **Assume array is sorted**
	☐ **If the array is full, call the resize function**
	☐ Insert the element into the correct position based on the value
		`Before:`
		`elem: [ 1, 3, 5, 10 ]`
		`count: 4`
		`max: 4`
		
		`insertSorted(&L, 8);`
		
		`After:`

		`elem: [ 1, 3, 5, 8, 10, ... ]`
		`count: 5`
		`max: 8`

`void display(List *L);`	☐ Display each element of the array until count is reached	
`void resize(List *L);`	☐ **Reallocate the memory assigned to the array and double its length** ☐ Double the max variable	
`void makeNULL(List *L);`	☐ Free the memory allocated to the List	

Note:

- For **INSERT FIRST** and **INSERT LAST**, you may use `insertPos`
- For **DELETE FIRST** and **DELETE LAST**, you may use `deletePos`
- For **DELETE BY VALUE**, you may use combination of `locate` and `deletePos`