# Data Structures

## Basic Abstract Data Types - List

Department of Software Engineering
College of Information Technology and Communications - Can Tho University

# Content

- Definition
- Oprerations of Lists
- Implementation of Lists
  - Array Implementation of Lists
  - Linked List

- A list is a sequence of zero or more elements of a given type (which we generally call the *elementtype*).
- We often represent such a list by a comma-separated sequence of elements: $a_1, a_2, a_3, \ldots, a_n$ where $n \geq 0$, and each $a_i$ is of type elementtype.
- The number $n$ of elements is said to be the length of the list.
- If
  - n = 0, we have an empty list, one which has no elements.
  - n ≥ 1, we say that $a_1$ is the first element and $a_n$ is the last element.
- Elements of list can be linearly ordered according to their position on the list. We say the element:
  - $a_i$ precedes $a_{i+1}$ for i = 1, 2, . . . , n-1,
  - $a_i$ follows $a_{i-1}$ for i = 2, 3, . . . ,n.
  - $a_i$ is at position $i$.

# Operations of Lists

| Operation | Use |
|---|---|
| **makenullList(L)** | Causes L to become an empty list. |
| **emptyList(L)** | Check if list L is empty or not. |
| **fullList(L)** | Check if list L is full or not. |
| **first(L)** | Returns the first position on list L. If L is empty, the position returned is endList(L). |
| **endList(L)** | Return the position following position n in an n-element list L. |
| **insertList(x,P,L)** | Insert x at position P in list L. If list L has no position p, the result is undefined. |
| **deleteList(P,L)** | Delete the element at position P of list L. The result is undefined if L has no position P or if P = endList(L). |

| Operation | Use |
|---|---|
| **retrieve(P,L)** | Returns the element at position P on list L. The result is undefined if P = endList(L) or if L has no position P. |
| **locate(x,L)** | Returns the position of x on list L. If x appears more than once, then the position of the first occurrence is returned. If x does not appear at all, then endList(L) is returned. |
| **next(P,L)** | Return the positions following position P on list L. If P is the last position on L, then next(P, L) = endList(L). next is undefined if P is endList(L). The result is undefined if L has no position P. |
| **previous(P,L)** | Return the positions preceding position P on list L. previous is undefined if P is 1. The result is undefined if L has no position P. |
| **printList(L)** | Print the elements of L in the order of occurrence. |

# Operations of Lists

Example: What operations do we call if we want to add an element x to the beginning or the end of the list?

- Add an element x to the beginning of the list

```
insertList(x, first(L), L)
```

- Add an element x to the end of the list

```
insertList(x, endList(L), L)
```

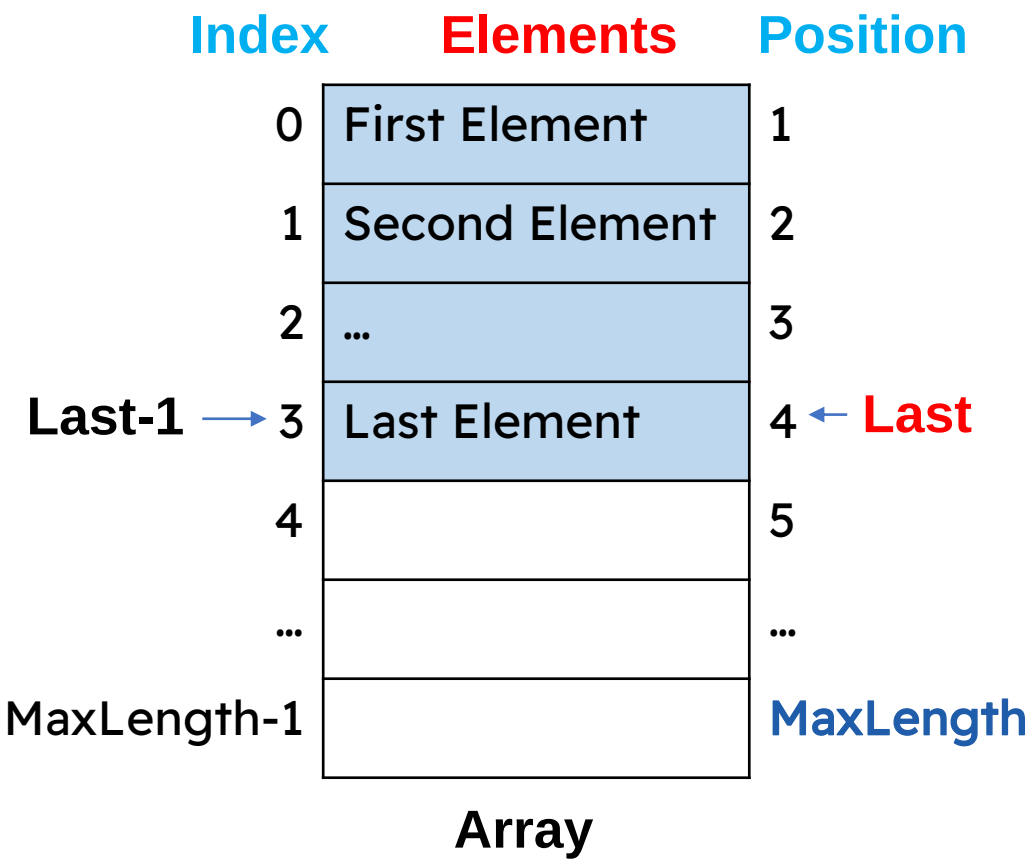
Example: Let the function *swap(p,q)* swap the contents of two elements at positions *p* and *q* in the list. Use operations on lists, write a function *sort* (where input parameter is a list *L*) to sort the list in ascending order?

```
void sort(List L){
        Position p,q;
        p= first(L);
        while (p!=endList(L)){
                q=next(p,L);
                while (q!=endList(L)){
                        if (retrieve(p,L) > retrieve(q,L))
                                swap(p,q);
                        q=next(q,L);
                }
                p=next(p,L);
        }
}
```

# Array Implementation of Lists (Array List)

- Declaration of data
- Declaration of operations

# Array Implementation of Lists (Array List)




- Use an array (Elements) to store contiguous elements, starting from the first position.
- Must estimate the maximum number of elements in the list (*MaxLength*).
- Must store the current length of the list (Last).

# Delaration of Data

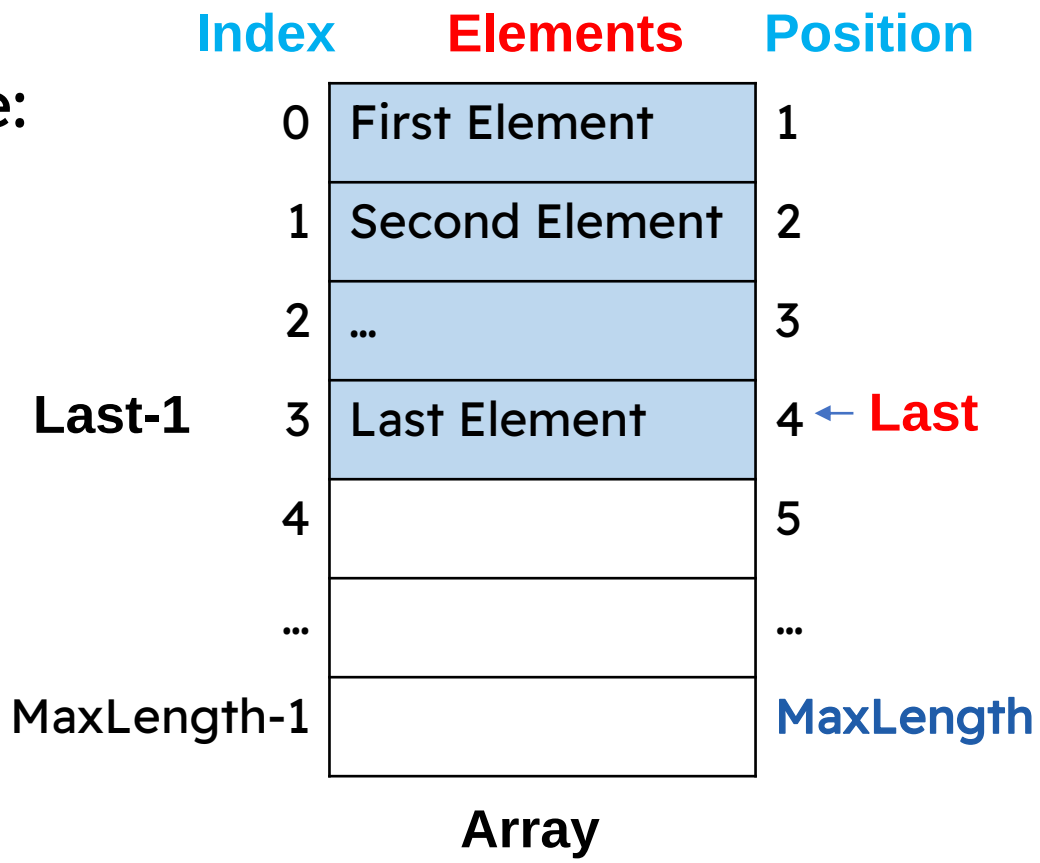
## A structure in C

- Declaring (creating, defining) a structure:

```
typedef struct
{
    <datatype>    <elementname1>;
    …
    <datatype>    <elementnamen>;
} <structurename>;
```

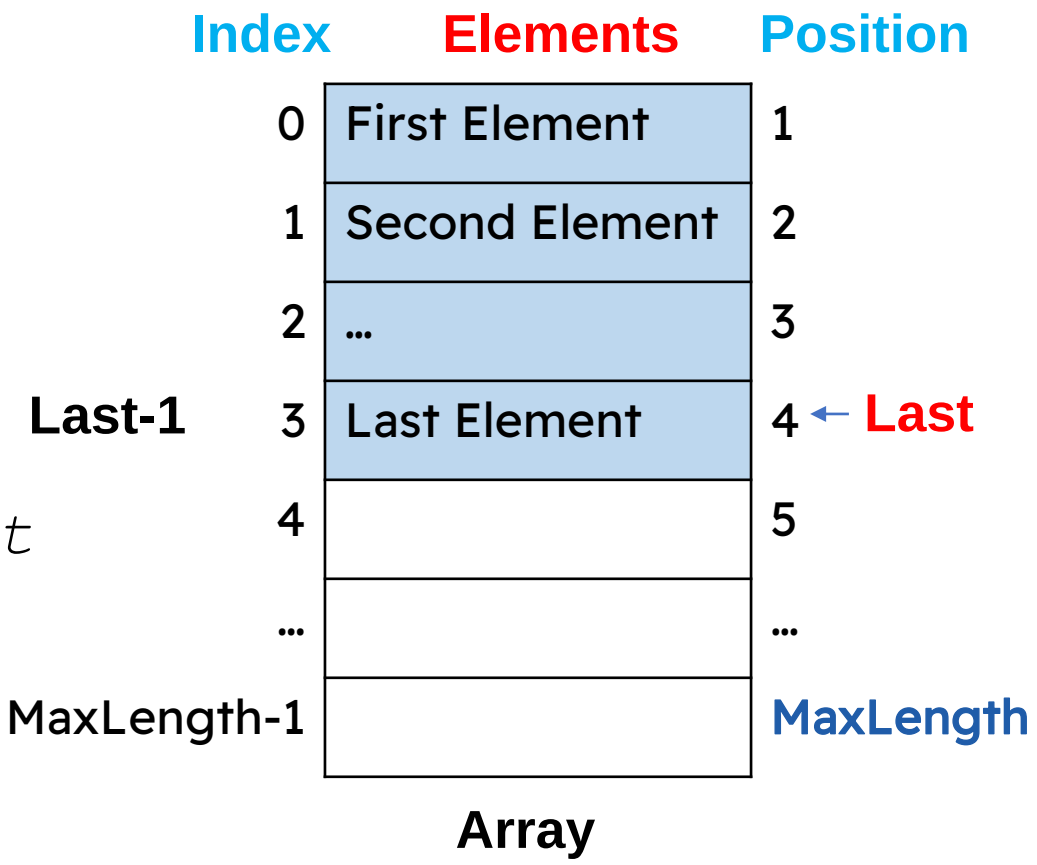
- Declaring a structure variable:

```
<structurename> <variablename>;
```



```
// the maximum size of list
#define MaxLength n
// the element type of list
typedef datatype ElementType;
// the position type of elements
typedef int Position;
typedef struct {
  // array consisting elements of list
  ElementType Elements[MaxLength];
  // the current length of list
  Position    Last;
} List;

List L;
```

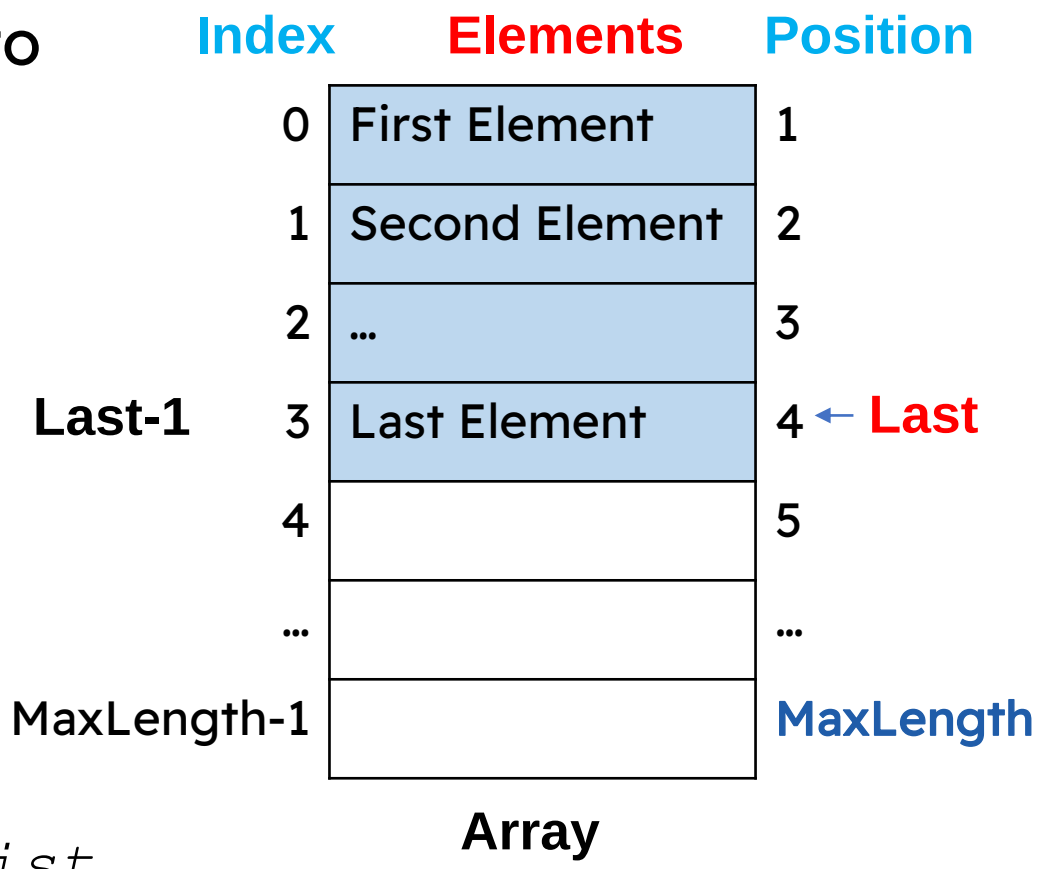



# Declaration of Data

Example: Declare a list that can contain up to 300 integers?

```
// the maximum size of list
#define MaxLength 300
// the element type of list
typedef int ElementType;
// the position type of elements
typedef int Position;
typedef struct {
  // array consisting elements of list
  ElementType Elements[MaxLength];
  // the current length of list
  Position    Last;
} List;

List L;
```




# Declaration of Data

# Exercise

- A line of text on the screen has a maximum of 80 characters. In fact, at a particular time the number of characters in the line is less than 80 so an integer variable is maintained to hold the actual number of characters in the line.
- To represent a line of text on the screen, we use an array list where each element of the line is a character.
  For example: 1 line of text

  *Toi di hoc. Ban o nha.*

  is recorded as a list with the following form.

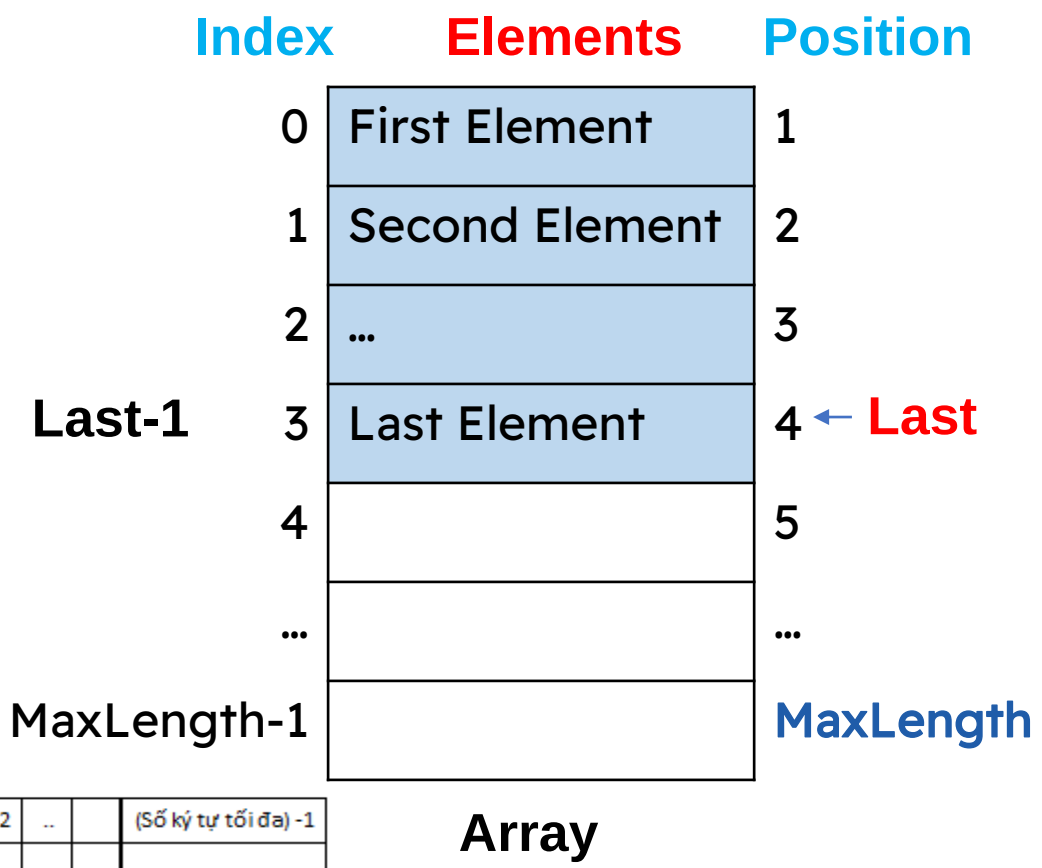
DÒNG VĂN BẢN

| *Chỉ số mảng* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | .. | | (Số ký tự tối đa) -1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Phần tử*** | **T** | **o** | **i** | | **d** | **i** | | **h** | **o** | **c** | **.** | | **B** | **a** | **n** | | **o** | | **n** | **h** | **a** | **.** | | | | |
| *Vị trí* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | ... | | (Số ký tự tối đa) |

***Số ký tự***

- Let's write a declaration for the text line data type Line (where the number of maximum characters is 80) with the elements described as shown above by using the array list.

```
// the maximum size of list
#define MaxLength n
// the element type of list
typedef datatype ElementType;
// the position type of elements
typedef int Position;
typedef struct {
   // array consisting elements of list
   ElementType Elements[MaxLength];
   // the current length of list
   Position    Last;
} List;
List L;
```



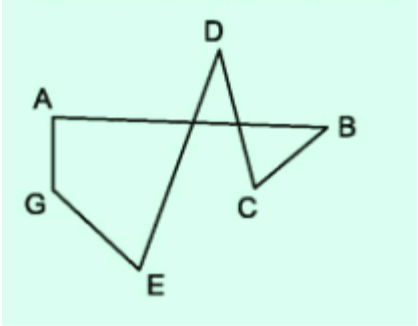

DÒNG VĂN BẢN

| Chỉ số mảng | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | .. | | (Số ký tự tối đa) -1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Phần tử | T | o | i | | d | i | | h | o | c | . | | B | a | n | | o | | n | h | a | . | | | | |
| Vị trí | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | ... | | (Số ký tự tối đa) |

Số ký tự

## Exercise

- A point in 2-dimensional space is described as a structure consisting of two x and y fields of integer type. A polygon is made up of many points, for example the polygon below is made up of 6 points: $A(x_A, y_A)$, $B(x_B,y_B)$, $C(x_C,y_C)$, $D(x_D,y_D)$, $E(x_E,y_E)$, $G(x_G,y_G)$.




- Suppose we represent a polygon by an array list with a maximum of 100 points. Write a declaration for the **Polygon** type representing polygons as described above.

# Declaration of Operations

- Defining a function

*Call by value* *Call by reference*

*Formal parameters (arguments)*

```
result_type  function_name(
            [parameter_type   parameter_name]
            [, [parameter_type    parameter_name][…])
{
      [declaration of local variables]
      [statements defining what the function does]
      [return [expression];]
}
```

- Calling a function in C

*Actual parameters*

```
function_name([a list of parameters])
```

# Declaration of Operations

*Call by value* *Call by reference*

*Formal parameters (arguments)*

- Defining a function

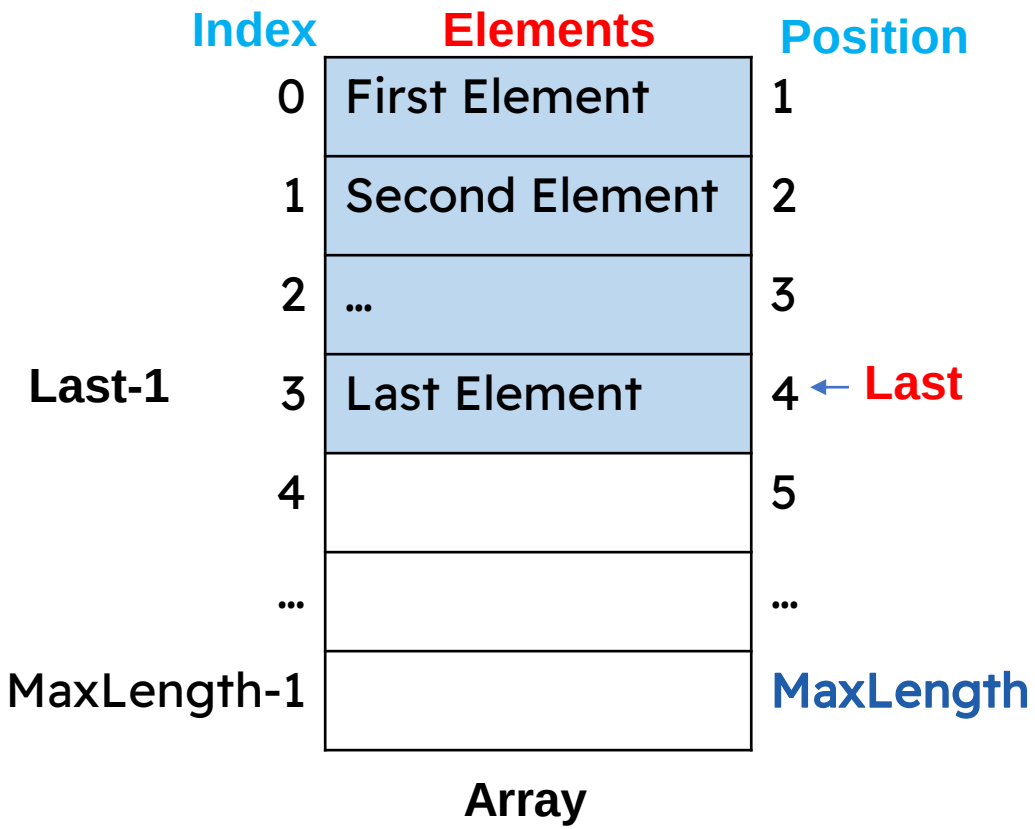
```
~~result_type~~  function_name(
   void        [parameter_type   parameter_name]
               [, [parameter_type    parameter_name][…])
{
      [declaration of local variables]
      [statements defining what the function does]
      [~~return [expression];~~]
}
```

- Calling a function

*Actual parameters*

```
function_name([a list of parameters])
```

# Declaration of Operations

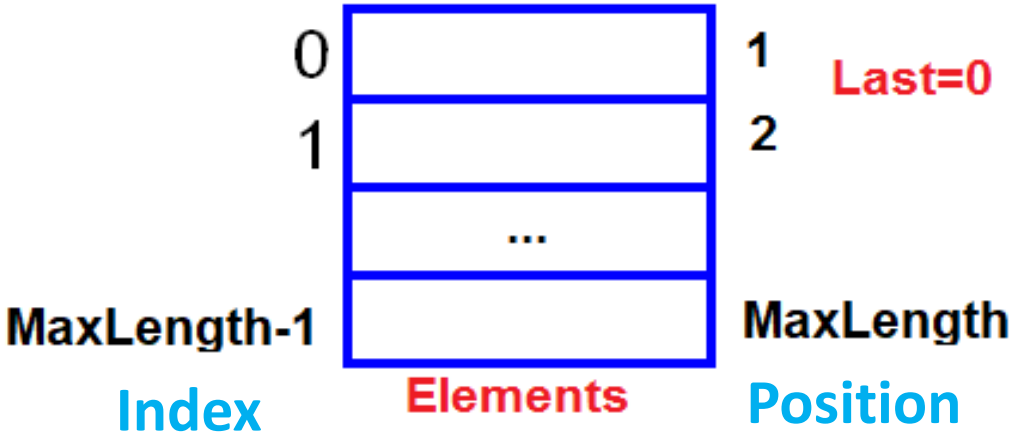



| Operator name |
|---|
| makenullList(L) |
| emptyList(L) |
| fullList(L) |
| first(L) |
| endList(L) |
| insertList(x,P,L) |
| deleteList(P,L) |
| retrieve(P,L) |
| locate(x,L) |
| next(P,L) |
| previous(P,L) |
| printList(L) |

```
result_type  function_name(
    [parameter_type   parameter_name]
    [, [parameter_type   parameter_name][…])
{
   [declaration of local variables]
   [statements defining what the function does]
   [return [expression];]
}
```

# Creating an Empty List

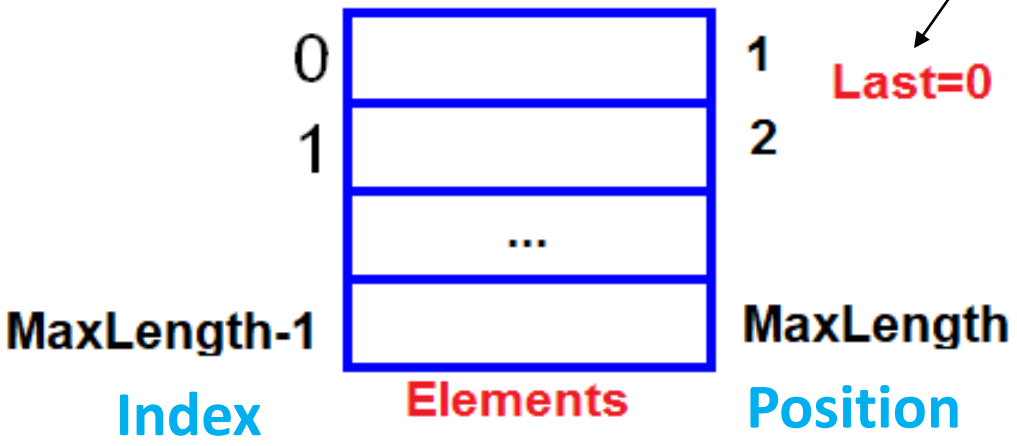
```
// the maximum size of list
#define MaxLength n
// the element type of list
typedef datatype ElementType;
// the position type of elements
typedef int Position;
typedef struct {
  // array consisting elements of
  ElementType Elements[MaxLength];
  // the current length of list
  Position    Last;
} List;
```




```
result_type  function_name(
    [parameter_type   parameter_name]
    [, [parameter_type   parameter_name][…])
{
  [declaration of local variables]
  [statements defining what the function does]
  [return [expression];]
}
```

# Creating an Empty List

Set the current length of list equals to 0

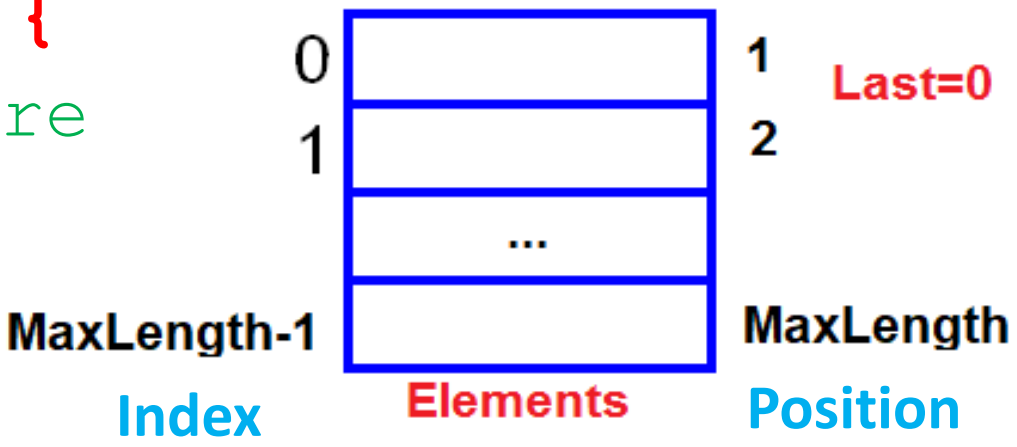



```
void makenullList(List *pL){
    // Pointer to structure
    pL->Last=0;
}
```

```
(*pL).Last=0
```

# Creating an Empty List

```
void makenullList(List *pL){
        // Pointer to structure
        pL->Last=0;
}
```



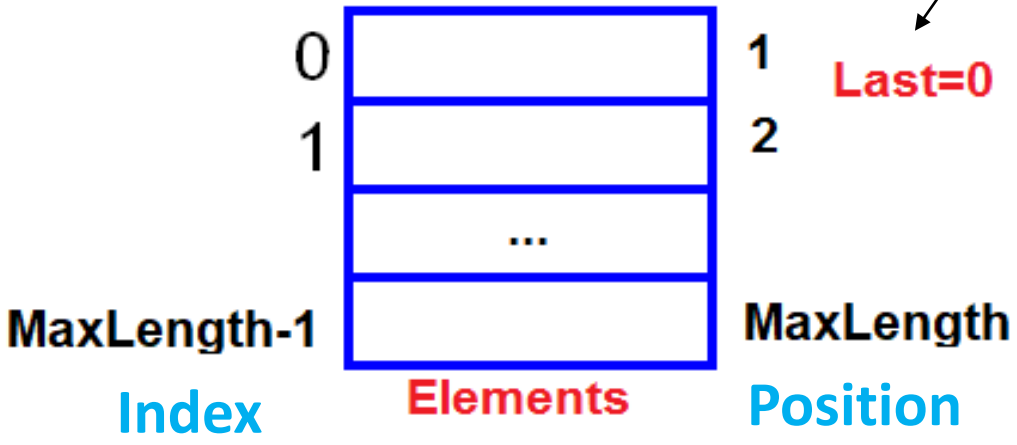

Suppose that we have a variable declaration:

```
List L;
```

Write a call to the `makenullList()` function to initialize an empty list `L`?

# Creating an Empty List

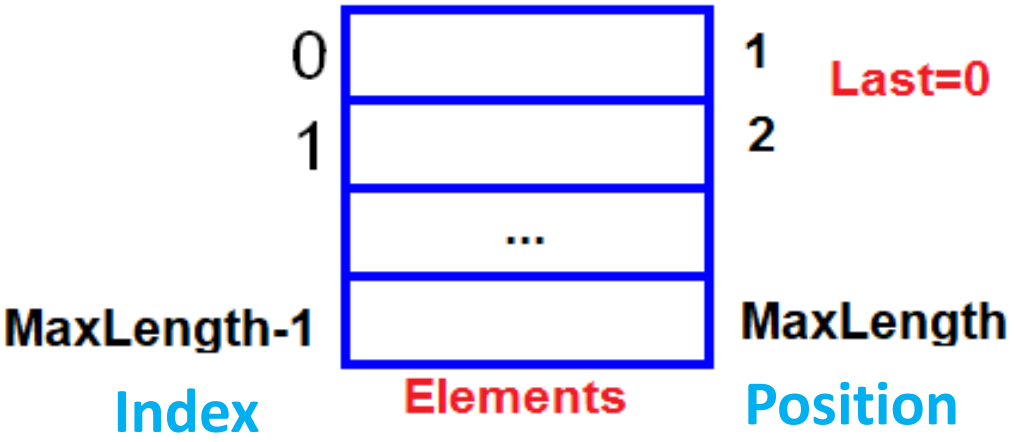
Set the current length of list equals to 0




- Write the function `makenullList` that takes no parameters and returns an empty list?

```
List makenullList(){
        List L; // L is a structure variable
        L.Last=0;
        return L;     }
```

# Creating an Empty List



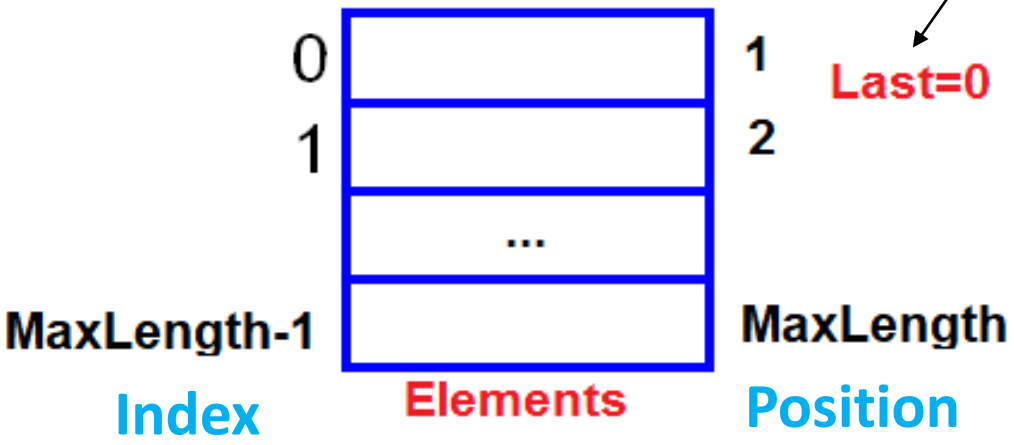

- Given the function to initialize an empty list.

```
List makenullList(){
        List L;
        L.Last=0;
        return L;      }
```

- Suppose that we have a variable declaration: `List L;`
- Write a call to the `makenullList()` function to initialize an empty list `L`?

# Checking an Empty List

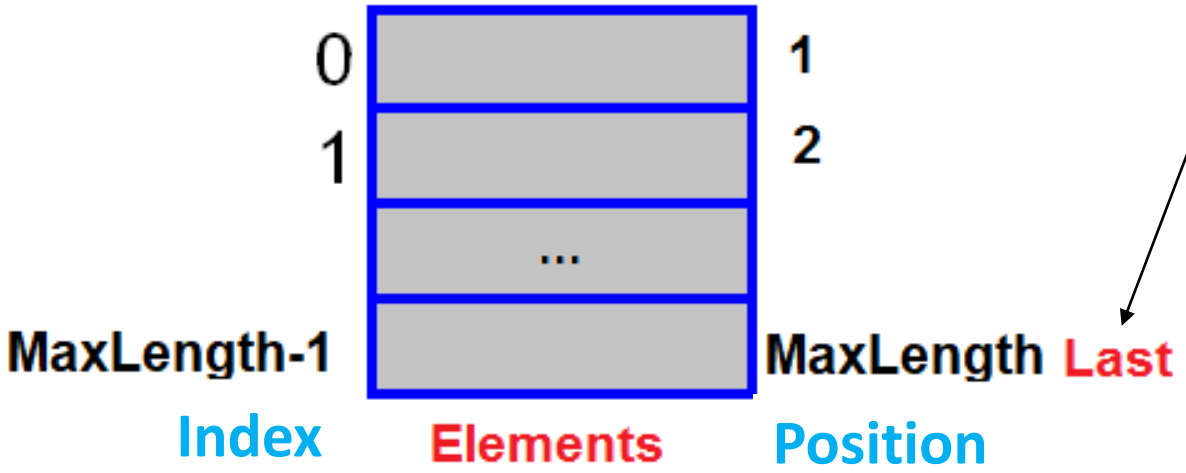
Check if the current length of list equals to 0




```
int emptyList(List L){
        return L.Last==0;
}
```

# Checking a Full List

Check if the current length of list equals to MaxLength

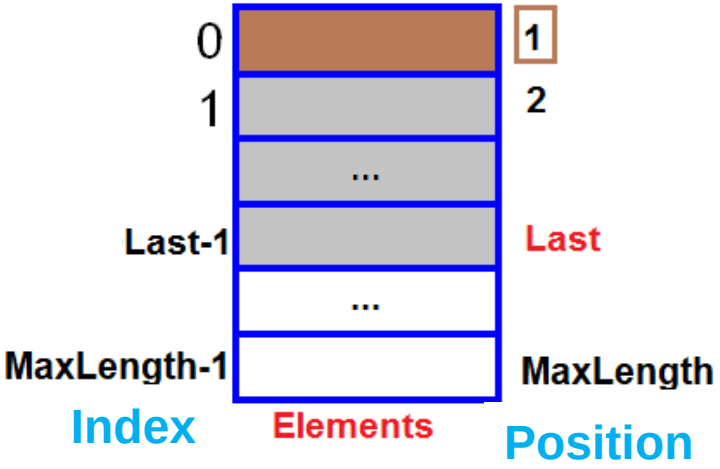



```
int fullList(List L)    {
        return L.Last==MaxLength;
}
```

# Determining Positions (first, endList)

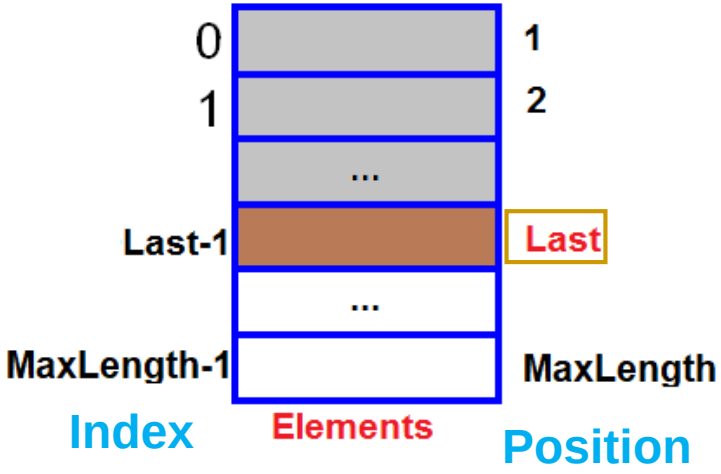
- Determining the first position of list.

```
Position first(List L){
        return 1;
}
```



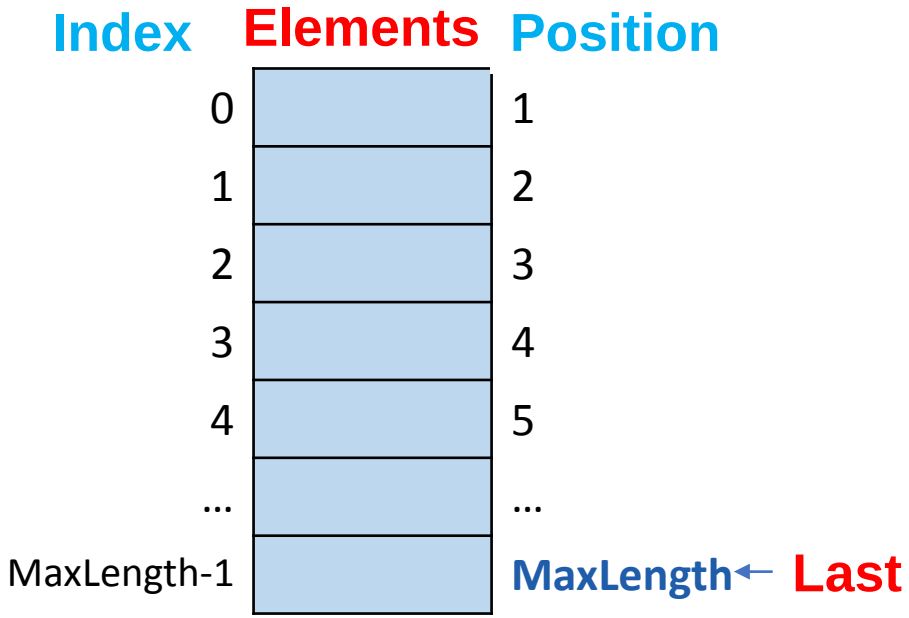

- Determining the position after the last element of list.

```
Position endList(List L){
   return L.Last+1;
}
```



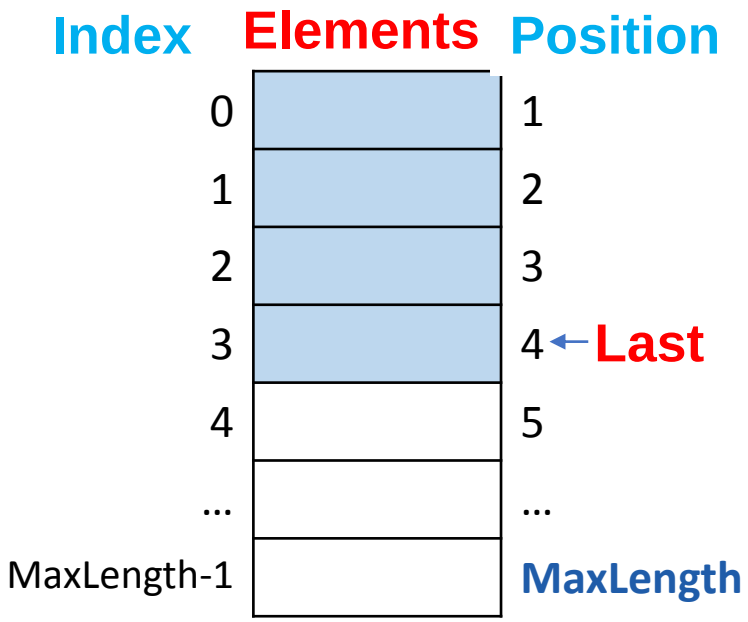
# Inserting an Element

- Example: insert the element x='k' into position P=3 in the list L?
  - Case 1: If L is the full list
    => show the error message



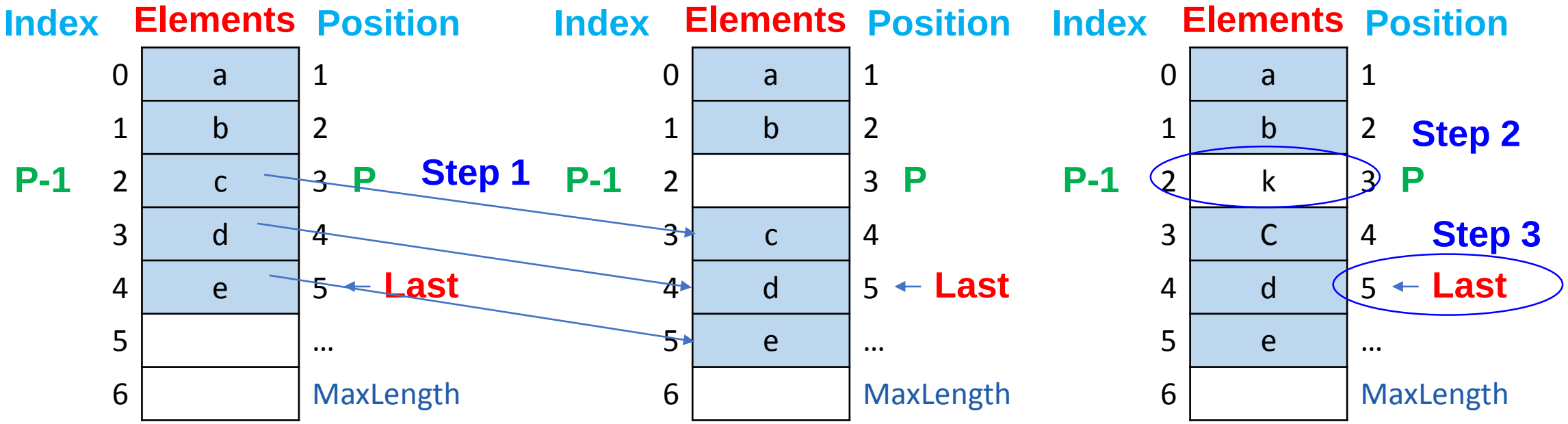
# Inserting an Element

- Example: insert the element x='k' into position P=3 in the list L?
  - Case 2: If position P is invalid (i.e. P∉[1..Last+1])
    => show the error message



# Inserting an Element

- Example: insert the element x='k' into position P=3 in the list L?
  - Case 3:




- Step 1: Moving elements at p and following positions (from position P to position Last) to the next higher position.
- Step 2: Inserting x at position P
- Step 3: Increasing the current length of list by 1.

# Inserting an Element

## Algorithm

To insert x into position P of the list L, we do the following:

- If the list is full, an error message is reported.
- Else,
  - If position P is invalid, an error message is reported.
  - Else:
    - Moving elements (from position P to position Last) to the next higher position.
    - Inserting new element x at position P.
    - Increasing the current length of list by 1.

```
void insertList(ElementType x,Position P, List *pL){
    if (pL->Last==MaxLength)
     printf("List is full");
    else if ((P<1) || (P>(pL->Last+1)))
       printf("Position is invalid");
    else {
       Position Q;
       for(Q=pL->Last;Q>=P;Q--)
        pL->Elements[Q]=pL->Elements[Q-1];
       pL->Elements[P-1]=x;
       pL->Last++;
    }
}
```

# Inserting an Element

```
void insertList(ElementType x,Position P, List *pL){
    if (pL->Last==MaxLength)
     printf("List is full");
    else if ((P<1) || (P>(pL->Last+1)))
       printf("Position is invalid");
    else {
       Position Q;
       for(Q=pL->Last;Q>=P;Q--)
        pL->Elements[Q]=pL->Elements[Q-1];
       pL->Elements[P-1]=x;
       pL->Last++;
    }
}
```

```
fullList(*pL)
```

```
(P<first(*pL) || (P>endList(*pL))
```

# Inserting an Element

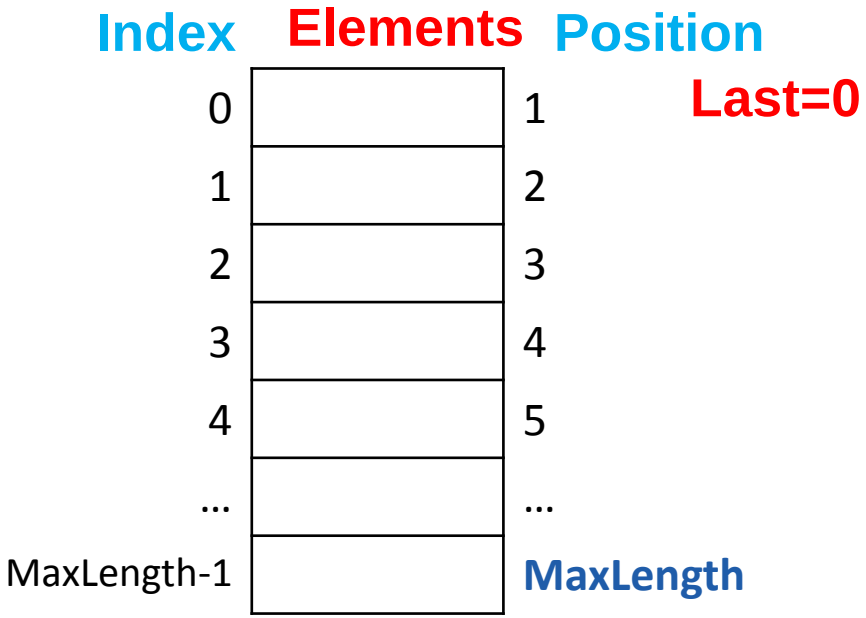
- Determining the complexity of insertList()?

# Inserting an Element

- Using operations on array lists, write the function `readList()` to input a list of `n` integers?

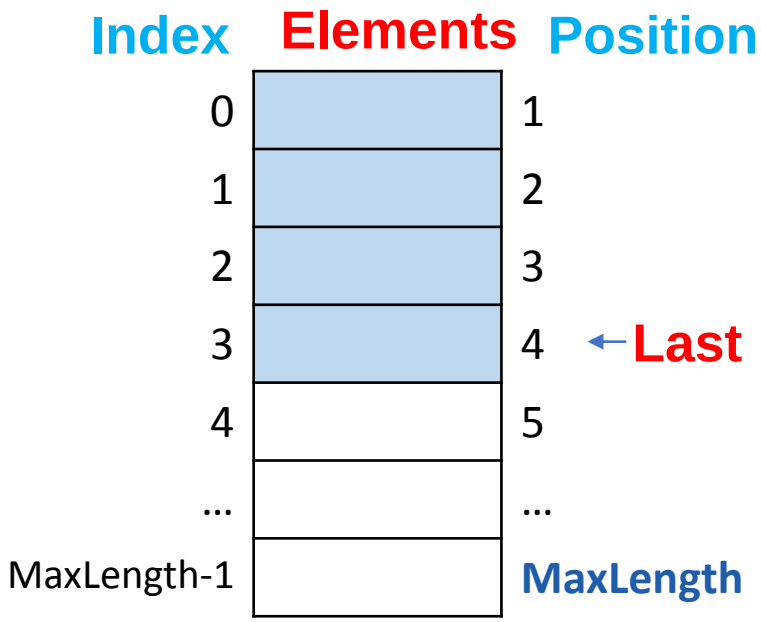
# Deleting an Element

- Example: delete the element at position P=4 of the list L?
  - Case 1: If L is the empty list => show the error message



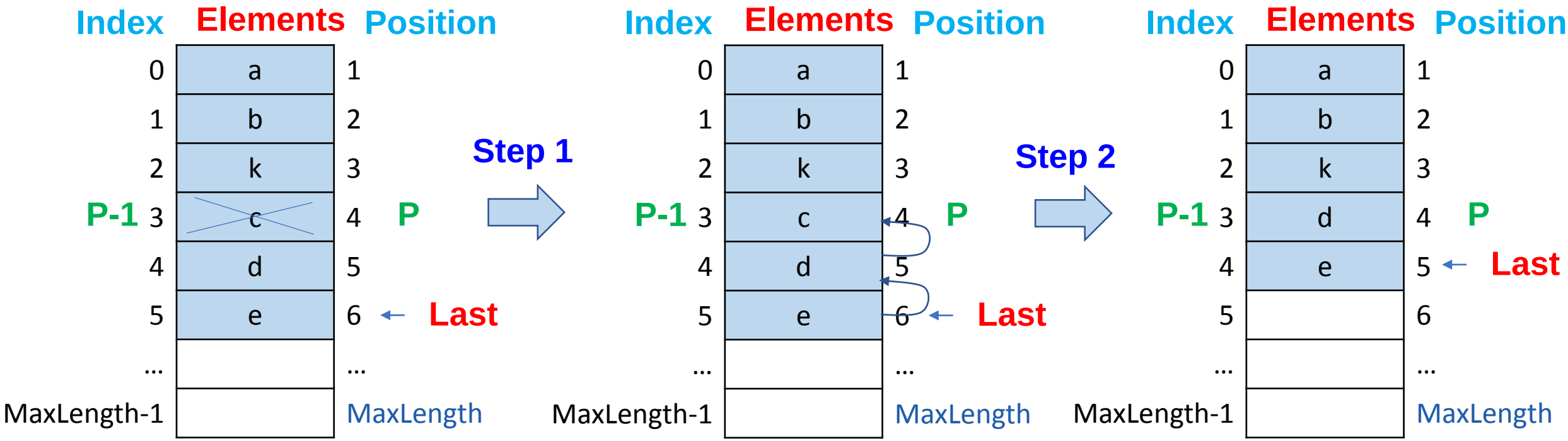
# Deleting an Element

- Example: delete the element at position P=4 of the list L?
  - Case 2: If position P is invalid (i.e. P∉[1..Last])
    => show the error message



- Example: delete the element at position P=4 of the list L?

  - Case 3:




- Step 1: Moving elements from position P+1 to position Last to the previous lower position.
- Step 2: Reducing the current length of list by 1.

# Deleting an Element

## Algorithm

To delete the element at position P of the list L, we do the following:

- If position P is invalid, an error message is reported.
- Else,
  - If the list is empty, an error message is reported.
  - Else:
    - Moving elements from position P+1 to position Last to the previous lower position.
    - Reducing the current length of list by 1.

```
void deleteList(Position P,List *pL){
  if ((P<1) || (P>pL->Last))
      printf("Position is invalid");
  else if (emptyList(*pL))
      printf("List is empty");
  else{
      Position Q;
     for(Q=P;Q<pL->Last;Q++)
         pL->Elements[Q-1]=pL->Elements[Q];
     pL->Last--;
  }
}
```

# Deleting an Element

# Deleting an Element

```
void deleteList(Position P,List *pL){
  if ((P<1) || (P>pL->Last))
      printf("Position is invalid");
  else if (emptyList(*pL))
      printf("List is empty");
  else{
      Position Q;
      for(Q=P;Q<pL->Last;Q++)
          pL->Elements[Q-1]=pL->Elements[Q];
      pL->Last--;
  }
}
```

pL->Last==0

```
for(Q=P-1;Q<pL->Last-1;Q++)
    pL->Elements[Q]=pL->Elements[Q+1];
```

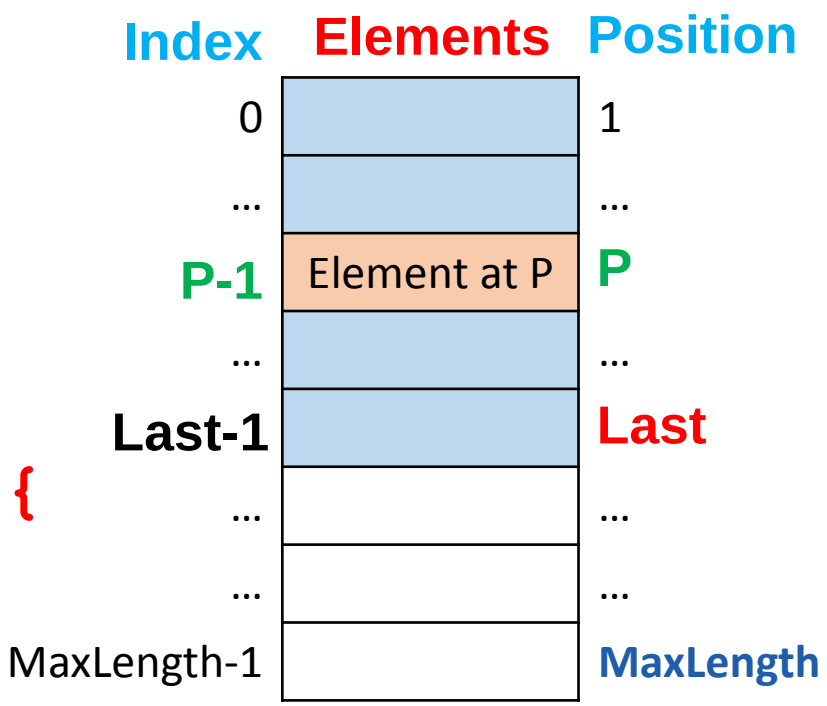
# Determining Positions (next, previous)

- Determining the previous position of P

```
Position next(Position P, List L){
        return P+1;
}
```

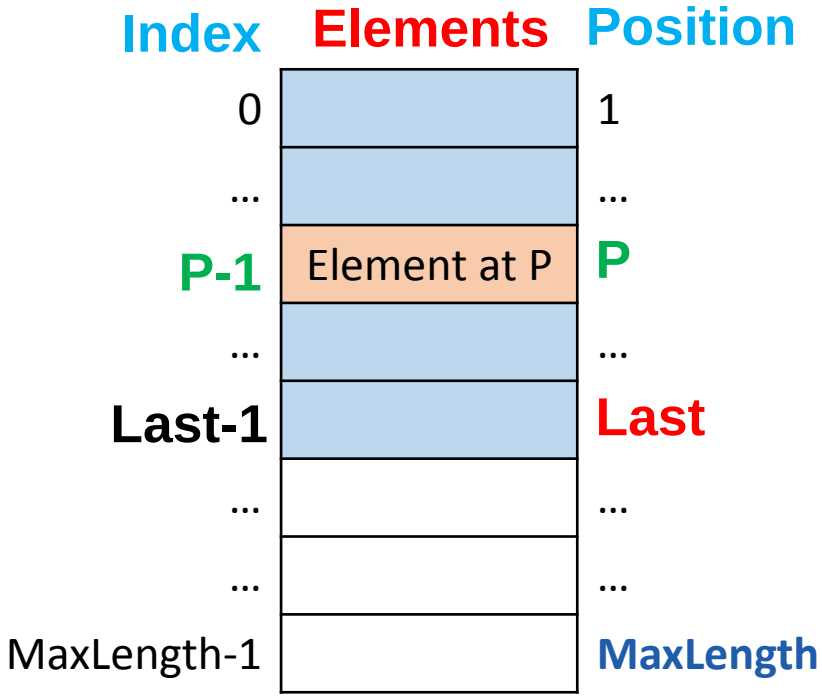
- Determining the next position of P

```
Position previous(Position P, List L){
   return P-1;
}
```



# Returning the Element

- Returning the element at position P of list L




```
ElementType retrieve(Position P, List L){
    return L.Elements[P-1];
}
```

# Searching the Element x

## Algorithm

To find the element x of the list L, we do the following:

- Starting from the first element of list, we search from the beginning of list until we find x or the end of list.
  - If the element at position P equals to x `retrieve(P,L) == x` then stop searching.
  - Else (if the element at position P differs from x), then go to the next position `P = next(P,L)`
- Returns the position of the first occurrence of x or `endList(L)` if x does not appear at all.

# Searching the Element x

```
Position locate(ElementType x, List L){
  Position P;
  int Found = 0;
  P = first(L);
  while ((P != endList(L)) && (Found == 0))
    if (retrieve(P,L) == x) Found = 1;
    else P = next(P, L);
  return P;
}
```

!Found

- Determining the complexity of locate ()?

# Searching the Element x

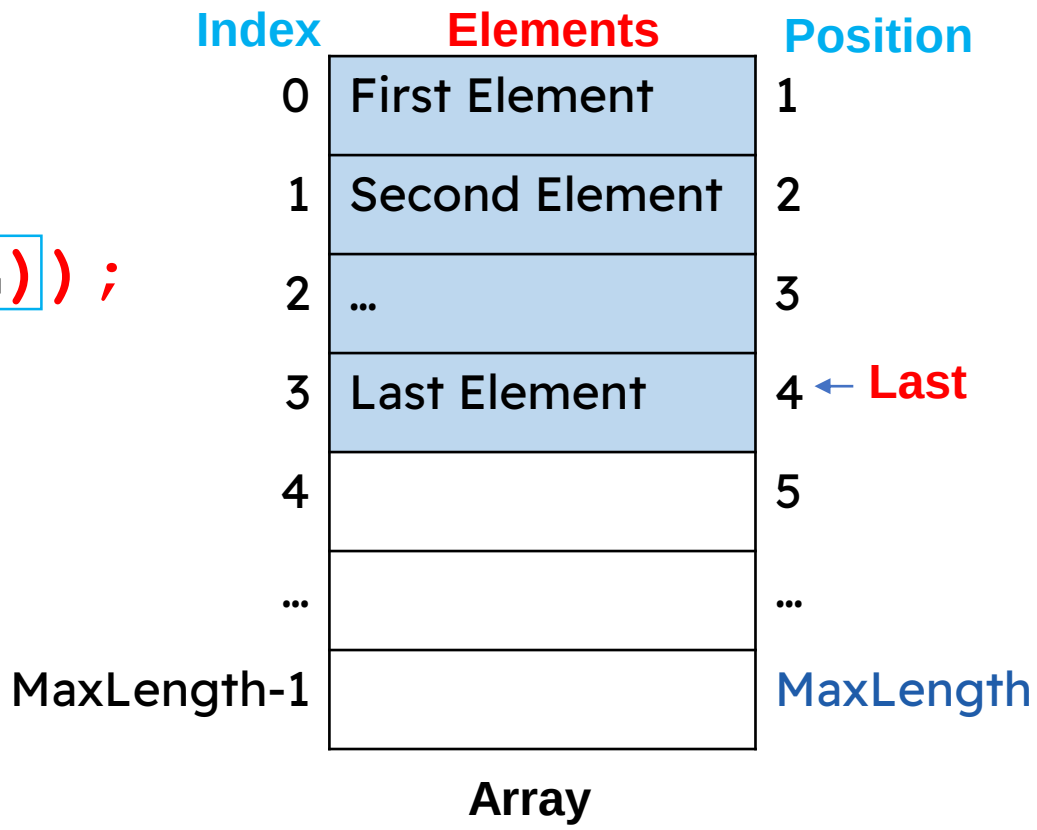
```
Position locate(ElementType x, List L){
  Position P;
  P = 1;
  while (P != L.Last+1)
    if ( L.Elements[P-1] == x) return P;
    else P = P+1;
  return P;
}
```

- Determining the complexity of locate ()?

# Printing the Elements

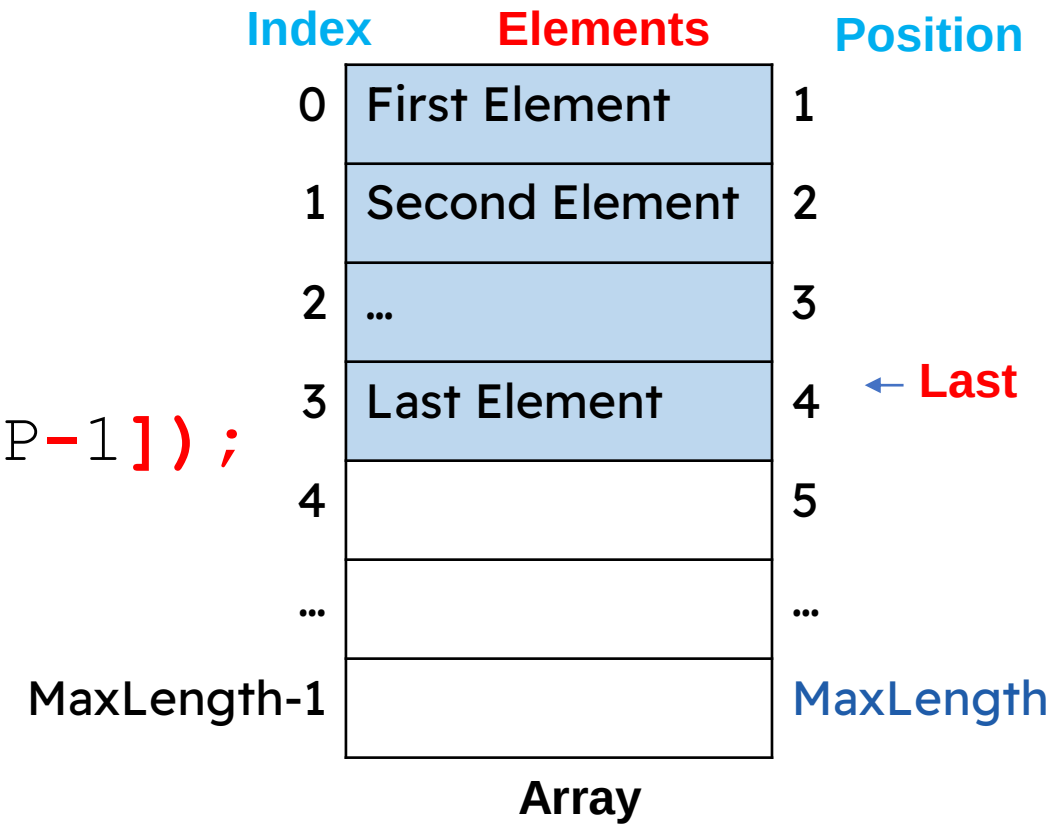
- Determining the complexity of printList ()?

```
void printList(List L){
    Position P = first(L);
    while (P != endList(L)){
        printf("%d ", retrieve(P,L));
        P = next(P,L);
    }
    printf("\n");
}
```



# Printing the Elements

```
void printList(List L){
   Position P = 1;
   while (P != L.Last+1){
        printf("%d ", L.Elements[P-1]);
        P = P+1;
   }
    printf("\n");
}
```



# Exercise

Using operations of array lists to write programs:

- Inputting a list of n integers and displaying that list on the screen.
- Inserting the element x at position P of the list. x and P are inputted from the keyboard.
- Deleting the first occurrence of x from the list. x is inputted from the keyboard.
- Writing the function `delete_duplicate(LIST &L)` that removes duplicate values in the list.

# Q&A