# ADT LIST

Abstract Datatype List

# DEFINITION

**Abstract Datatype**

- is a conceptual model of a data structure that is defined by its behavior, or the set of operations that can be performed on it, rather than its concrete implementation.
- It separates the **WHAT** (the functionality) from the **HOW** (the implementation).

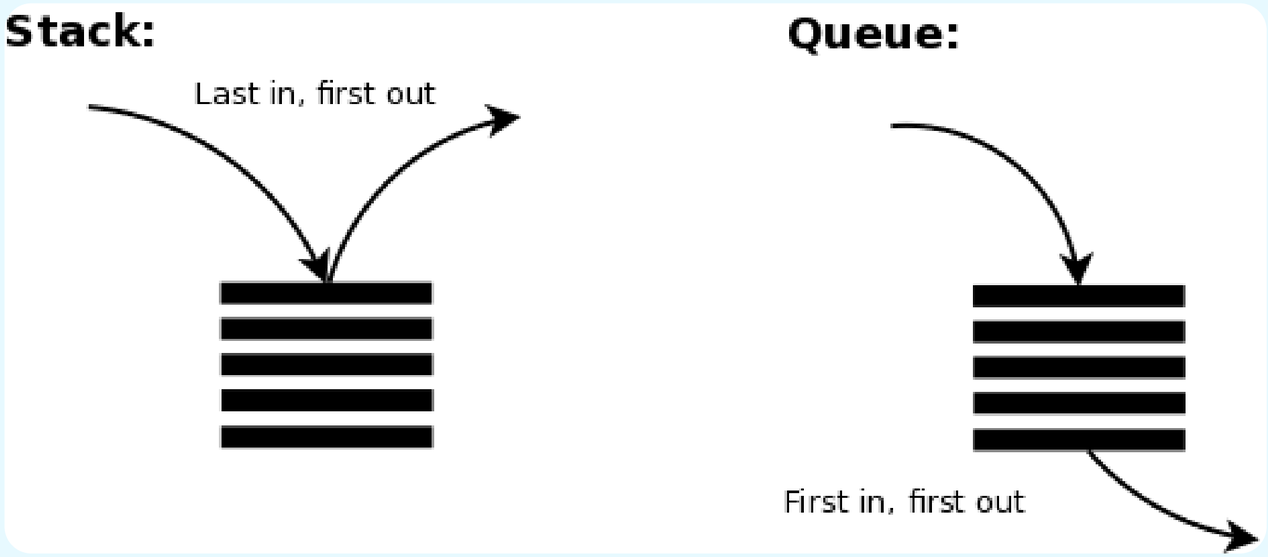
**Encapsulation** - Uses structs to hold the data and related functions to manipulate it.

**Abstraction** - The user only sees a simplified, high-level view of the data type.

# DEFINITION

## Abstract Datatypes

Ex. Lists, Stacks, Queue



# ABSTRACT DATATYPES

**Implementations**

Array

Linked List

Cursor Based

**Common Operations**

- Insert
- Sort
- Delete/Remove
- Update
- Initialize
- Search/Locate
- Display/Retrieve
- Make Null/Empty

# 4 VARIATIONS OF LIST

**Variation 1: LIST** is a list that is static array and accessed by value

**Variation 2: LIST** is a list that is static array and accessed by pointer

**Variation 3: LIST** is a list that is dynamic array and accessed by value

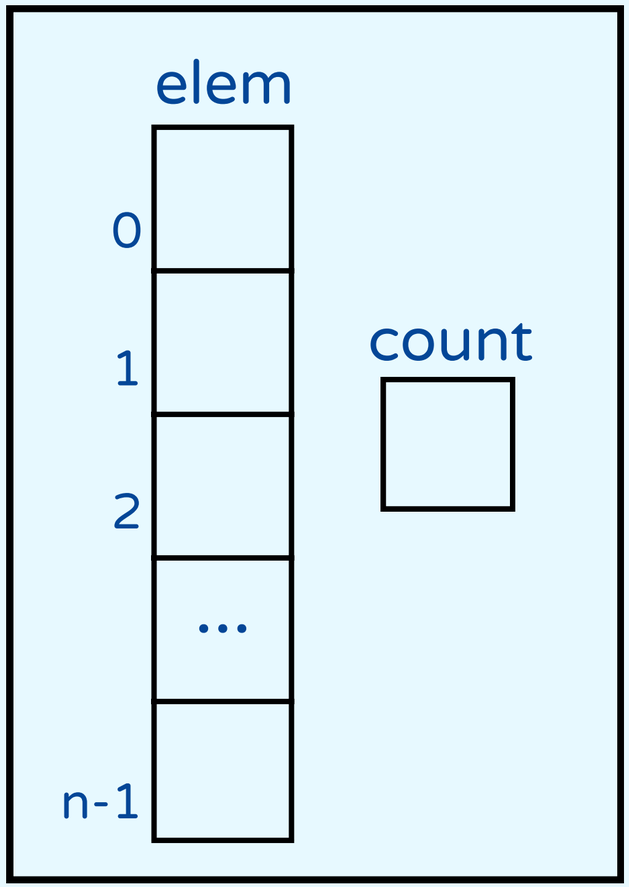
**Variation 4: LIST** is a list that is dynamic array and accessed by pointer

# 4 VARIATIONS OF LIST

| | Memory Management | Structure Access | Scalability & Flexibility |
|---|---|---|---|
| **Variation 1: Static, by Value** | • Statically allocated at compile time<br>• The size of the array is fixed and cannot be changed | • A copy of the entire list is made when passed to a function or assigned to a new variable | • Not scalable or flexible |

# VARIATION 1 LIST

List L




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

List L;
```

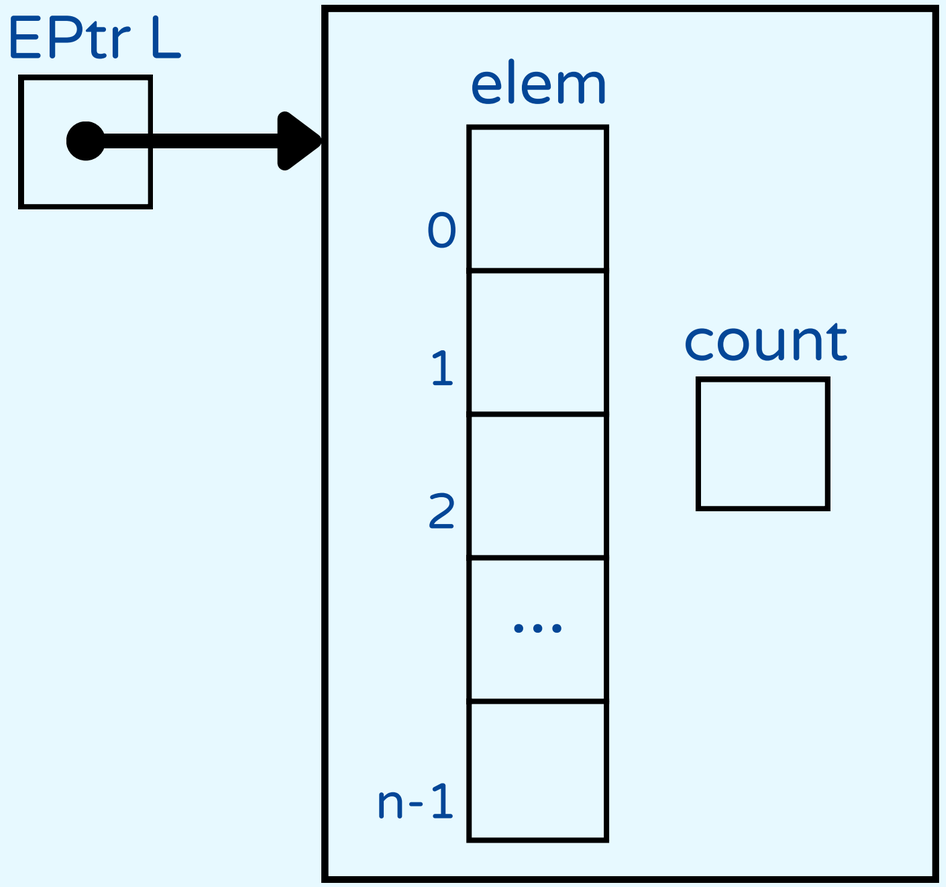
**Function Prototypes:**

- List initialize (List L);
- List insertPos (List L, int data, int position);
- List deletePos(List L, int position);
- int locate (List L, int data);
- List insertSorted (List L, int data);
- void display(List L);

# 4 VARIATIONS OF LIST

| | Memory Management | Structure Access | Scalability & Flexibility |
|---|---|---|---|
| **Variation 2: Static, by Pointer** | • Statically allocated at compile time<br>• The size of the array is fixed and cannot be changed | • Only the address is passed, not a full copy | • Not scalable or flexible |

# VARIATION 2 LIST




```
#define MAX 10

typedef struct{
    int elem[MAX];
    int count;
} Etype, *EPtr;

EPtr L;
```

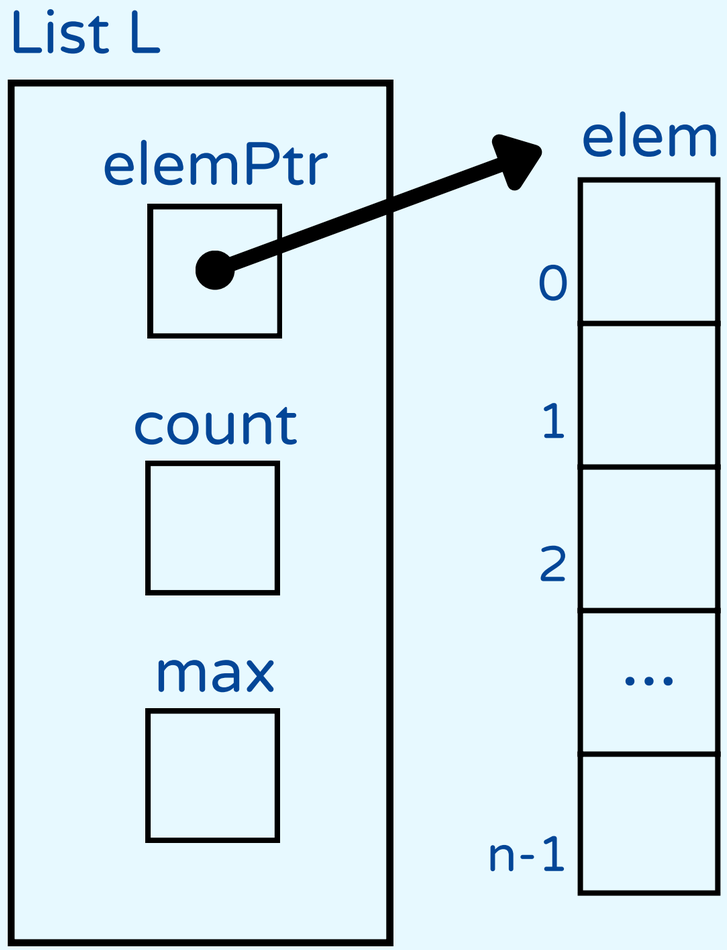
**Function Prototypes:**

- void initialize (EPtr L);
- void insertPos (EPtr L, int data, int position);
- void deletePos(EPtr L, int position);
- int locate (EPtr L, int data);
- int retrieve (EPtr L, int position);
- void insertSorted (EPtr L, int data);
- void makeNULL(EPtr L);

# 4 VARIATIONS OF LIST

| | Memory Management | Structure Access | Scalability & Flexibility |
|---|---|---|---|
| **Variation 3: Dynamic, by Value** | • Dynamically allocated at run-time<br>• The size can be changed as needed | • A copy of the entire list is made | • Scalable but not flexible<br>• The list can grow or shrink in size. However, creating a new copy for every operation is inefficient |

# VARIATION 3 LIST




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

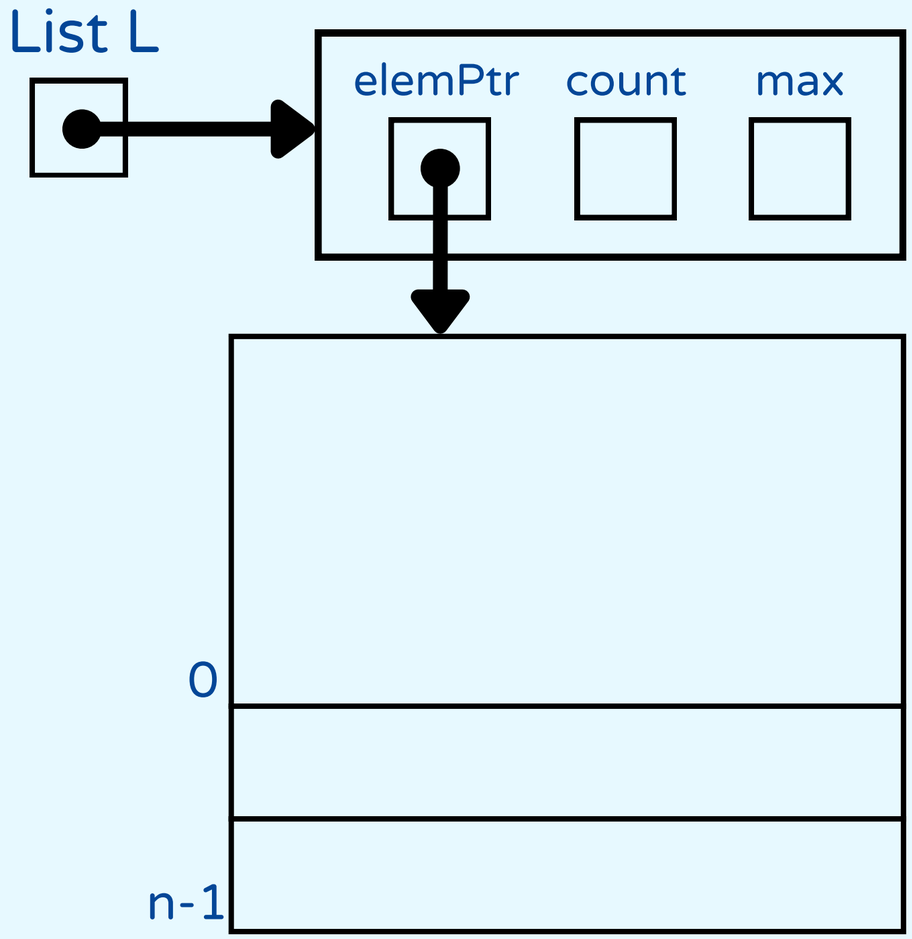
**Function Prototypes:**

- List initialize (List L);
- List insertPos (List L, int data, int position);
- List deletePos(List L, int position);
- int locate (List L, int data);
- List  insertSorted (List L, int data);
- void display(List L);
- List resize(List L);

# 4 VARIATIONS OF LIST

| | Memory Management | Structure Access | Scalability & Flexibility |
|---|---|---|---|
| **Variation 4: Dynamic, by Pointer** | • Dynamically allocated at run-time<br>• The size can be changed as needed | • Only the address is passed, not a full copy | • Highly scalable and flexible<br>• Since access is by pointer, only a small memory address is passed, making operations efficient |

# VARIATION 4 LIST




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

**Function Prototypes:**

- void initialize (List *L);
- void insertPos (List L, studtype elem, int pos);
- void deletePos(List L, int pos);
- void locate (List L, int ID);
- studtype retrieve (List L, int pos);
- void insertSorted (List L, studtype elem);
- void makeNULL(List *L);
- void resize(List L);

# THANK YOU