Data Structures and Algorithms
(ES221)

List

Dr. Zubair Ahmad

- **Attendance?**

  - Active Attendance
  - **Dead Bodies.**
  - **Active Minds**
  - Mobiles in hands -> Mark as absent
  - 80% mandatory

Primitive Data Types

Name	Description	Size	Range
char	Character or small integer	1 byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer	2 bytes	signed: -32768 to 32767 unsigned: 0 to 65535
Int	Integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
iong int (long)	Long integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false	1 byte	true or false
float	Floating point number	4 bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number	8 bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number	8 bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	Wide character	2 or 4 bytes	1 wide character

Abstract Data Types

A **mathematical model** for a class of **data structures** with similar behavior.

Defines **operations** without specifying implementation details.

Focuses on **what** the data structure does, **not how** it is implemented.

Examples:

- **Lists, Sets, Graphs** (as ADTs).

Abstract Data Types

ADT is a **set of operations** without specifying implementation.

:

•**Write once, reuse multiple times**.
•Access ADT functions without knowing internal details.

:

•Changes in implementation affect only ADT functions.
•Rest of the program remains **unaffected** (transparent changes).

Abstract Data Types

Simplifies **algorithm design** and classification of **data structures**.
Helps define **type systems** in programming languages.

Can be implemented using **various data structures** across multiple languages.
May be formally described in a **specification language**

Abstract Data Types: Example

Example: abstract stack (functional)
A complete functional-style definition of a stack ADT could use the three operations:

push: takes a stack state and an arbitrary value, returns a stack state;
top: takes a stack state, returns a value;
pop: takes a stack state, returns a stack state;

List

The ***list*** *data type* is a collection type that stores ordered, non-unique elements; that is, it allows duplicate element values.

Operations

Traverse through the list
- search for an element in the list
- read/update an element in the list (at the beginning, end, anywhere)

Delete an element from the list
- delete from the beginning of the list
- delete from the end of the list
- delete any element in the list

Add a new element in the list
- add at the beginning of the list
- add at the end of the list
- add/insert anywhere in the list

List: Traverse through the list

Visit each element of the list

e.g., print each element of the list on the screen

List data	2	4	6	8	10	12	14	16

```
for (int i = 0; i < N; i++)
cout<<a[i];
```

Complexity : O(N)

List: Search for an element in the list

Search for a specific element in the list

e.g., search and return the index of 10, if found in the list

List data	2	4	6	8	10	12	14	16

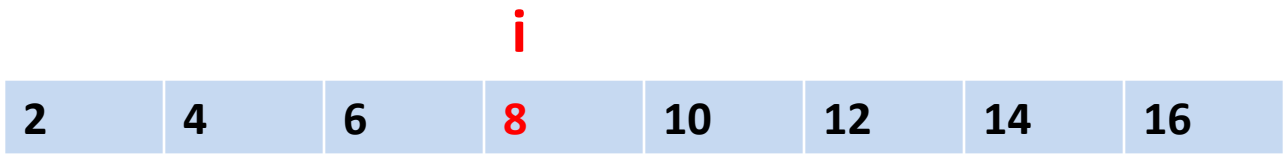
```
SearchElement = 10;
 for (int i = 0; i < N; i++){
            if (a[i] = = SearchElement){
                index = i;
                 return index;
             }
 }
index = -1;
cout<< "Element not found";
return index;
```

Complexity : O(*N*)

List: Read/Update an Element

- **Read/update an element in the list**
  - at the beginning,
  - at the end
  - anywhere
  - Example
    - Read/print *i*th element (*i*=3)
    - update *j*th element (*j*=4)

Read *i*th element



$i = 3;\ cout << a[i];$ → **8**

List: Read/Update an Element

Update *i*th element

j

2	4	6	8	10	12	14	16

2	4	6	8	15	12	14	16

$j = 4;\ a[j] = 15;$

Complexity : O(1)

List: Add Element

Add at the beginning:

Step 1 : Move all the elements towards right, creating space for one more element in the beginning

Step 2 : Add the new element at the beginning

Step 3 : Increment the size of the array by one

```
 N = N + 1;
    for (int i = N-1; i > 0; i--
)
          a[i] = a[i-1];
      a[0] = NewElement; // (=
1)
```

Complexity : O(*N*)

List: Add Element

Add at the end:

Step 1 : Add the new element at end

Step 2 : Increment the size of the array by one

$$N = N + 1;$$
$$a[N\text{-}1] = NewElement \text{ ; // } (= 17)$$

Complexity : O(*1*)

List: Add Element

Add anywhere in the list (e.g., at the *j*th location)

Step 1 : Move all the elements, starting from the index *j*, towards right,

Step 2 : Add the new element at the index *j*

Step 3 : Increment the size of the array by one

```
N = N +1;
    for (int i = N-1; i > j; i-
-)
            a[i] = a[i-1];
            a[j] = NewElement ;
```

Complexity : O(*N*)

List: Delete Element

Delete the beginning element:

Step 1 : Move all the elements towards left, overriding the first element
Step 2 : Decrement the size of the array by one

```
for (int i = 0; i < N-1; i++)
        a[i] = a[i +1];
        N = N - 1;
```

Complexity : O(N)

List: Delete Element

Add at the end:

Step 1 : Delete the element from the end

Step 2 : Decrement the size of the array by one

Initial data

2	4	6	8	10	12	14	16

Delete element and decrement size

2	4	6	8	10	12	14

Complexity : O(1)

$N = N - 1;$

List: Delete Element

Delete from anywhere in the list (e.g., at the *j*th location)

Step 1 : Move all the elements, after the index *j*, towards left

Step 2 : Decrement the size of the array by one

```
for (int i = j; i < N-1; i++)
            a[i] = a[i+1];
            N = N -1;
```

Complexity : O(*N*)

List: Search and Update a Specific Element

```
void searchAndUpdate(int a[], int N, int target, int newValue)
{
    for (int i = 0; i < N; i++) {
        if (a[i] == target) {  // Found the target element
            a[i] = newValue;   // Update the value
            cout << "Updated element " << target << " to " <<
newValue << " at index " << i << endl;
            return;
        }
    }
    cout << "Element " << target << " not found!" << endl;
}
```

Complexity : O(N)

List: Insert After a Specific Element

Steps

1.Search for the element (AfterElement).
2.Shift elements right from j+1.
3.Insert new element at j+1.
4.Increment N

Complexity : O(N)

List: Insert After a Specific Element

```
void insertAfter(int a[], int N, int
AfterElement, int newElement) {
    int j = -1;

for (int i = 0; i < N; i++) {
        if (a[i] == AfterElement) {
            j = i;
            break;
        }
    }

    if (j == -1) {
        cout << "Element " <<
AfterElement << " not found!" << endl;
        return;
    }
```

```
for (int i = N; i > j + 1; i--)
        a[i] = a[i - 1];

    // Step 3: Insert at j+1
    a[j + 1] = newElement;
    N++;  // Step 4: Increment size

    cout << "Inserted " << newElement
<< " after " << AfterElement << endl;
}
```

List: Insert Before a Specific Element

Steps

1. **Search** for the element (BeforeElement).
2. **Shift elements right** from j.
3. **Insert new element** at j.
4. **Increment N**

List: Insert Before a Specific Element


```
void insertBefore(int a[], int &N, int
BeforeElement, int newElement) {
    int j = -1;

    // Step 1: Find BeforeElement
    for (int i = 0; i < N; i++) {
        if (a[i] == BeforeElement) {
            j = i;
            break;
        }
    }

    if (j == -1) {
        cout << "Element " <<
BeforeElement << " not found!" << endl;
        return;
    }

    // Step 2: Shift elements right from j
    for (int i = N; i > j; i--)
        a[i] = a[i - 1];

    // Step 3: Insert at j
    a[j] = newElement;
    N++;  // Step 4: Increment size

    cout << "Inserted " << newElement << "
before " << BeforeElement << endl;
}
```

List: Insert Before a Specific Element

Time Complexity

Search for BeforeElement???

Shift elements to the right ???

Insert new element at the correct position ??

Update the array size ???

Overall Complexity: O(N) (because of shifting)

List – Array Implementation Challenges

The cost of Inserting an element anywhere inside the list is $O(N)$
The cost of deleting an element anywhere from the list is $O(N)$

Can we reduce this cost to $O(1)$?

List implemented as an dynamic array

The static declaration of an array requires the size of the array to be known in advance

What if the actual size of the list exceeds its expected size?

Solution?

- Dynamic array
- Linked List

Assignment!!

Details of Assignment will be shared today anytime before 5 pm on the course webpage. Please follow them and submit accordingly

Questions?

zahmaad.github.io