Objectives (CLO1, CLO3)

ArrayList ADT (List Using Arrays) in C++

An **ArrayList ADT (Abstract Data Type)** is a **resizable list** built **using arrays**. In simple words, it behaves like an array but can **add, remove, update, and access** elements easily. Even though arrays in C++ have **fixed size**, we can create a **custom list** that:

- Uses an **array internally**,
- Tracks its **current size**,
- Performs operations like insert, delete, search, etc.

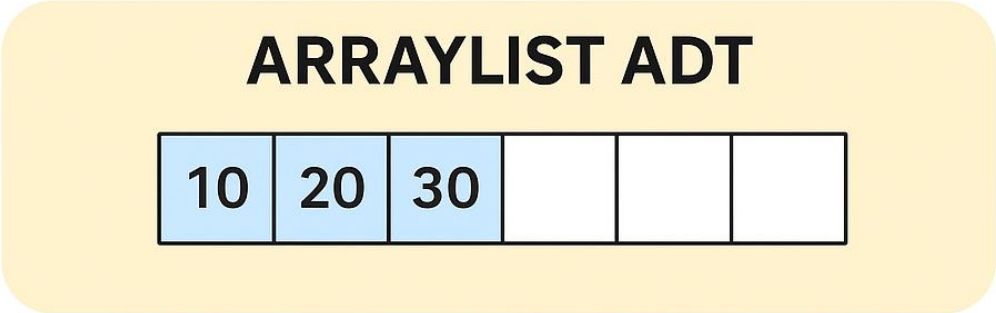


Figure 1 arraylist ADT

Core components of an array-based list ADT

A custom class for an array-based list typically needs the following member variables:

- A pointer to a dynamic array, e.g., T* data. This array stores the actual list elements.
- An integer to track the current number of elements in the list, e.g., currSize.
- An integer to store the total capacity of the underlying array, e.g., capacity

Why We Need ArrayList ADT

Normal Array	ArrayList ADT
Fixed in size	Grows or shrinks
Manual resizing needed	Automatic resizing logic
Basic operations	Provides add, remove, get, set, etc.
Harder to manage	Easier and reusable

Key operations and their implementation

A C++ class implementing an array-based list ADT includes several standard functions to manage the list elements.

Constructor

The constructor initializes the list with a default capacity and sets the current size to zero**.**

```
template <typename T>
class ArrayList {
private:
    T* data;
    int currSize;
    int capacity;

public:
    ArrayList() {
        capacity = 10; // Start with a default capacity
        data = new T[capacity];
        currSize = 0;
    }
    // Other methods...
};
```

Destructor

The destructor is essential for preventing memory leaks by deallocating the memory reserved for the dynamic array.

```
    ~ArrayList() {
        delete[] data;
    }
```

add(item)

To add an element to the end of the list, first check if the current size has reached the capacity. If so, call a helper function to increase the array's size (e.g., by doubling it). Then, add the new element and increase the currSize.

```
    void add(T item) {
```

```
    if (currSize == capacity) {
        resize();
    }
    data[currSize++] = item;
}
```

add(index, item)

To insert an element at a specific index, you must first shift all subsequent elements to the right to make space. This is an $O(n)$operation where n is the number of elements.

```
void add(int index, T item) {
    if (index < 0 || index > currSize) {
        // Handle error: index out of bounds
        return;
    }
    if (currSize == capacity) {
        resize();
    }
    // Shift elements to the right
    for (int i = currSize; i > index; --i) {
        data[i] = data[i - 1];
    }
    data[index] = item;
    currSize++;
}
```

get(index)

Accessing an element at a specific index is a fast, constant-time operation $O(1)$ because arrays provide direct, random access to their elements.

```
T get(int index) {
    if (index < 0 || index >= currSize) {
        // Handle error: index out of bounds
        return T(); // Return a default value or throw an exception
    }
```

```
    return data[index];
}
```

remove(index)

Removing an element at a given index requires shifting all subsequent elements to the left to close the gap. Like insertion, this is an $O(n)$ operation.

```
void remove(int index) {
    if (index < 0 || index >= currSize) {
        // Handle error: index out of bounds
        return;
    }
    // Shift elements to the left
    for (int i = index; i < currSize - 1; ++i) {
        data[i] = data[i + 1];
    }
    currSize--;
}
```

resize()

When the array is full, this private helper function creates a new, larger array. It copies all elements from the old array to the new one, deletes the old array, and updates the pointer and capacity.

```
void resize() {
    capacity *= 2; // Double the capacity
    T* newData = new T[capacity];
    for (int i = 0; i < currSize; ++i) {
        newData[i] = data[i];
    }
    delete[] data;
    data = newData;
}
```

Problem Statement:

Write a C++ program to **implement an ArrayList Abstract Data Type (ADT)** using **arrays**. The program should allow the list to **store integers dynamically**, performing these operations:

1. **Add** elements to the end of the list.
2. **Insert** an element at a specific index (shifting other elements to the right).
3. **Remove** an element from a specific index (shifting elements left).
4. **Access (get)** an element by its index.
5. **Resize** the array automatically when it becomes full (double its capacity).
6. **Print** all elements of the list.

Demonstrate all these operations in the main() function.

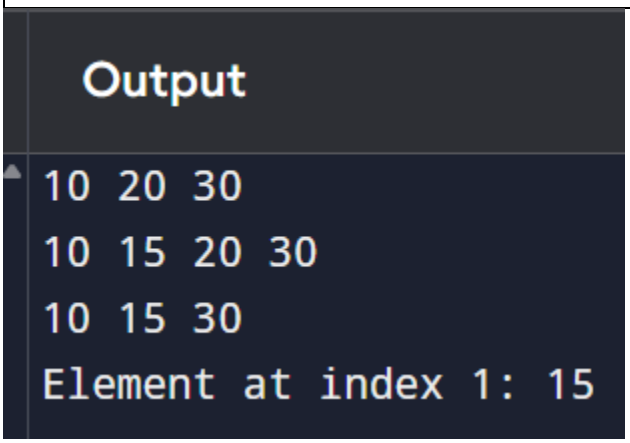
Implementation

```
#include <iostream>
using namespace std;

class ArrayList {
private:
    int* data;
    int size;
    int capacity;

public:
    ArrayList(int cap = 5) {
        capacity = cap;
        size = 0;
        data = new int[capacity];
    }

    void add(int value) {
        if (size == capacity) {
            cout << "List is full, resizing...\n";
            resize();
        }
        data[size++] = value;
    }
```

```
void insert(int index, int value) {
    if (index < 0 || index > size) {
        cout << "Invalid index!\n";
        return;
    }
    if (size == capacity) resize();

    for (int i = size; i > index; i--)
        data[i] = data[i - 1];
    data[index] = value;
    size++;
}

void remove(int index) {
    if (index < 0 || index >= size) {
        cout << "Invalid index!\n";
        return;
    }
    for (int i = index; i < size - 1; i++)
        data[i] = data[i + 1];
    size--;
}

int get(int index) {
    if (index < 0 || index >= size) {
        cout << "Invalid index!\n";
        return -1;
    }
    return data[index];
}
```

```
    void print() {
        for (int i = 0; i < size; i++)
            cout << data[i] << " ";
        cout << endl;
    }

    void resize() {
        int newCap = capacity * 2;
        int* newData = new int[newCap];
        for (int i = 0; i < size; i++)
            newData[i] = data[i];
        delete[] data;
        data = newData;
        capacity = newCap;
    }
};

int main() {
    ArrayList list;

    list.add(10);
    list.add(20);
    list.add(30);
    list.print();

    list.insert(1, 15);
    list.print();

    list.remove(2);
    list.print();
```

```
    cout << "Element at index 1: " << list.get(1) << endl;
    return 0;
}
```



Practice Questions

1. Write a C++ program using your **ArrayList ADT** that allows the user to: Add several integer elements. Enter a value to **search** for in the list. If found, **delete** that element and display the updated list. If not found, display a message: *"Element not found!"*

Hint: Use a loop to check each element and use your remove(index) function to delete the matched value.

2. Write a C++ program using your **ArrayList ADT** that: Stores a list of integers entered by the user. Counts how many numbers are **even** and how many are **odd**. Displays the count of each category.

Hint: raverse the ArrayList using a loop and use the condition:if (data[i] % 2 == 0) even++; else odd++;