DATA STRUCTURES & ALGORITHMS

LISTS

issntt@nus.edu.sg

A problem

Using arrays, write an app that **stores players** in **teams**

```
class Player
{
   public string Name {
      set; get; }
   public int Number {
      set; get; }

   public Player(
     string name, int number)
   {
      Name = name;
      Number = number;
   }

   public override
         string ToString() {
      return Name + " "
                 + Number;
   }
}
```

```
class Team {
   public string TeamName {
      set; get; }

   // Add necessary attributes

   public void AddPlayer(
              Player player) {
      // To implement
   }

   public void RemovePlayer(
              Player player) {
      // To implement
   }

   public void Print() {
      // To implement
   }
}
```

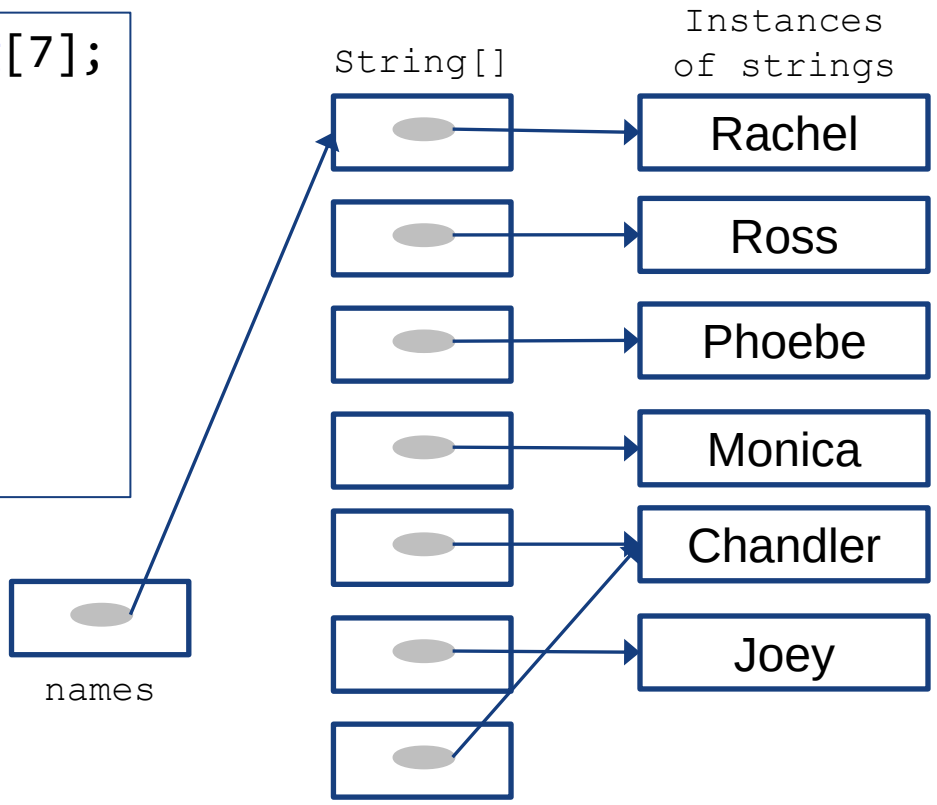
Outline

- **A review of Arrays**
  - Players-Team solution using Arrays
  - Problems with using Arrays
- What are Abstract Data Types (ADT)?
- List ADT
- Using List ADT
- Implementing List ADT

Arrays

An array stores a **fixed-size sequential** collection of **elements** of the **same type**

```
string[] names = new string[7];
names[0] = "Rachel";
names[1] = "Ross";
names[2] = "Phoebe";
names[3] = "Monica";
names[4] = "Chandler";
names[5] = "Joey";
names[6] = names[4];
```



Accessing Elements

An array **element** is **accessed using** its respective **index** and array indexes **start from 0**, not 1

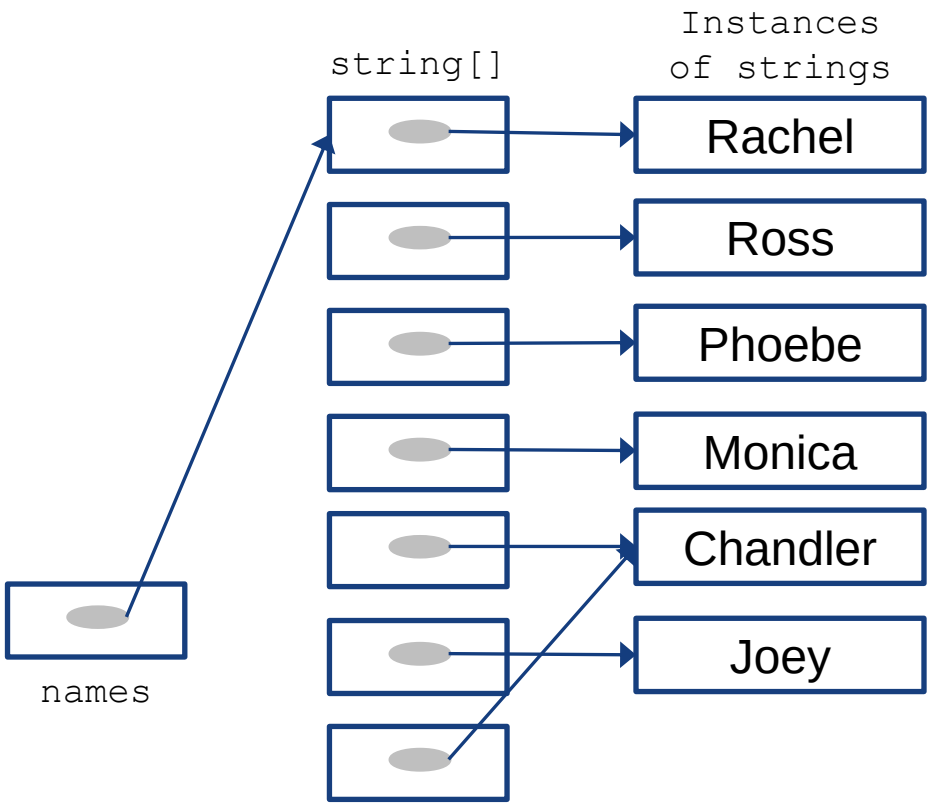
```
static void ArrayExample() {
    string[] names = new string[7];
    names[0] = "Rachel";
    names[1] = "Ross";
    names[2] = "Phoebe";
    names[3] = "Monica";
    names[4] = "Chandler";
    names[5] = "Joey";
    names[6] = names[4];

    Console.WriteLine(names[0]);
    Console.WriteLine(names[6]);
}
```

```
Rachel
Chandler
```

Accessing Elements

Accessing an array element with index is **very fast**

How does the computer access *names[6]*? Does it go through *names[0]*, *names[1]*... first?



Outline

- **A review of Arrays**
  - **Players-Team solution using Arrays**
  - Problems with using Arrays
- What are Abstract Data Types (ADT)?
- List ADT
- Using List ADT
- Implementing List ADT

A Players-Team solution

Use an **array** to **keep** the **players**, and another variable to **keep** the **number** of **players**

```
class Team
{
  private const int MAX_CAPACITY = 10;
  public string TeamName { set; get; }

  private Player[] players;
  private int numPlayers;

  public Team(string teamName)
  {
    players = new Player[MAX_CAPACITY];
    numPlayers = 0;

    TeamName = teamName;
  }
```

Adding Players

When adding, use the variable ***numPlayers*** as **index** to add to the **end** of the array

```
public void AddPlayer(Player player)
{
    players[numPlayers] = player;
    numPlayers++;
}
```

Removing Players

When removing, **find** the respective **index** and **make** the respective object ***null***

```
public void RemovePlayer(Player player) {
    int index = RetrievePlayerIndex(player);
    if (index != -1) {
        players[index] = null;
    }
}
private int RetrievePlayerIndex(Player playerToRetrieve) {
    for (int i = 0; i < players.Length; i++) {
        if (players[i] != null &&
              players[i].Name.Equals(playerToRetrieve.Name) &&
              players[i].Number == playerToRetrieve.Number) {
            return i;
        }
    }
    return -1;
}
```

Printing Players

When printing, **loop** through the array and **print** out the **non-null elements**

```
public void Print()
{
    Console.WriteLine(TeamName);
    for (int i = 0; i < players.Length; i++)
    {
        if (players[i] != null)
        {
            Console.WriteLine(players[i]);
        }
    }
}
```

Testing Application

Let's test our program

```
static void Main(string[] args){
    Team team = new Team("Dream Team");
    team.AddPlayer(new Player("Yashin", 1));
    team.AddPlayer(new Player("Beckenbauer", 5));
    team.AddPlayer(new Player("Messi", 10));
    team.AddPlayer(new Player("C. Ronaldo", 7));
    team.AddPlayer(new Player("Ronaldo", 9));

    team.RemovePlayer(new Player("Yashin", 1));

    team.Print();
}
```

```
Dream Team
Beckenbauer 5
Messi 10
C. Ronaldo 7
Ronaldo 9
```

Question

What **issues** may be with this solution?

Image by GraphicMama-team from Pixabay

Outline

- **A review of Arrays**
  - Players-Team solution using Arrays
  - **Problems with Arrays**
- What are Abstract Data Types (ADT)?
- List ADT
- Using List ADT
- Implementing List ADT

Issues with the array approach

To solve many problems better, we need

- **Better tools**
- Thinking from the **perspective** of the **tools**

Image by free stock photos from www.picjumbo.com from Pixabay

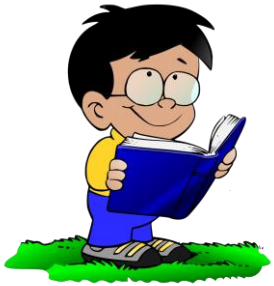
Outline

- A review of Arrays
- **What are Abstract Data Types (ADT)?**
- List ADT
- Using List ADT
- Implementing List ADT

Question

Do you care **how things work inside ATMs** when **withdrawing money**?

Image by Peggy und Marco Lachmann-Anke from Pixabay



What is Abstraction?

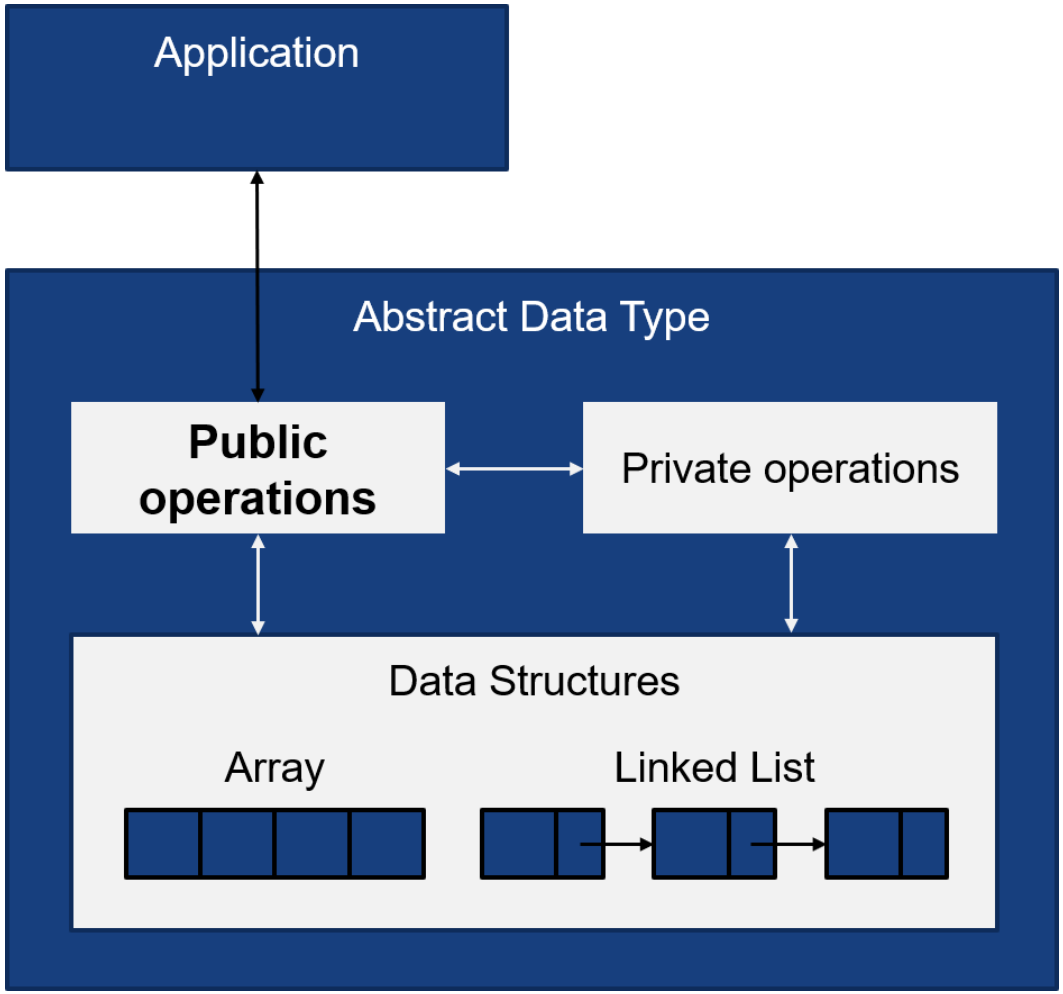
The act of **distilling** a **complicated system** down to its **most fundamental parts**

Inside an ATM. Source: Bjoertvedt/Wikimedia

Abstract Data Type (ADT)

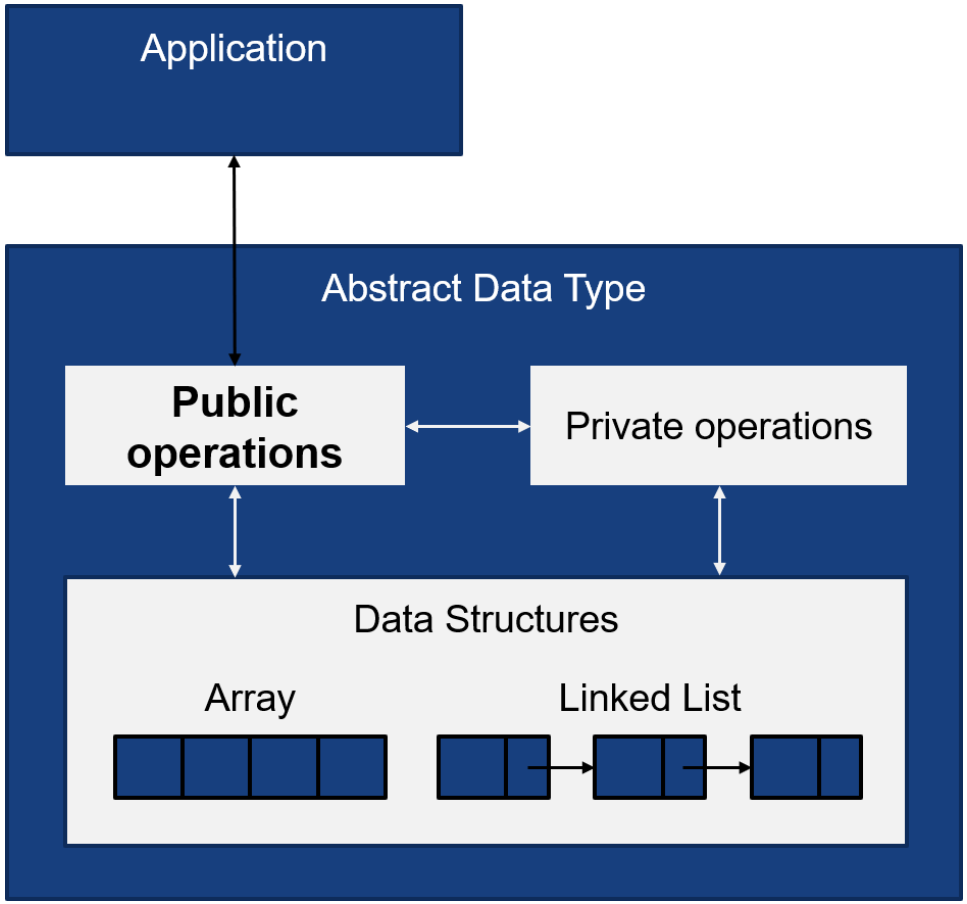
A mathematical **model** of a **data type** that specified:

- The **public operations supported** on them
- The **types of parameters** on the operations



Abstract Data Type (ADT)

ADT is from the **point of view** of **users**, specifying **what** each public operation does, **not how** it does it



Does the application care about the **private operations** and the underlying **data structures**?

Outline

- A review of Arrays
- What are Abstract Data Types (ADT)?
- **List ADT**
- Using List ADT
- Implementing List ADT

List

- A list is an **ADT** that **stores** a **collection** of **items** in a **sequential order**
- May contain **duplicate entries**

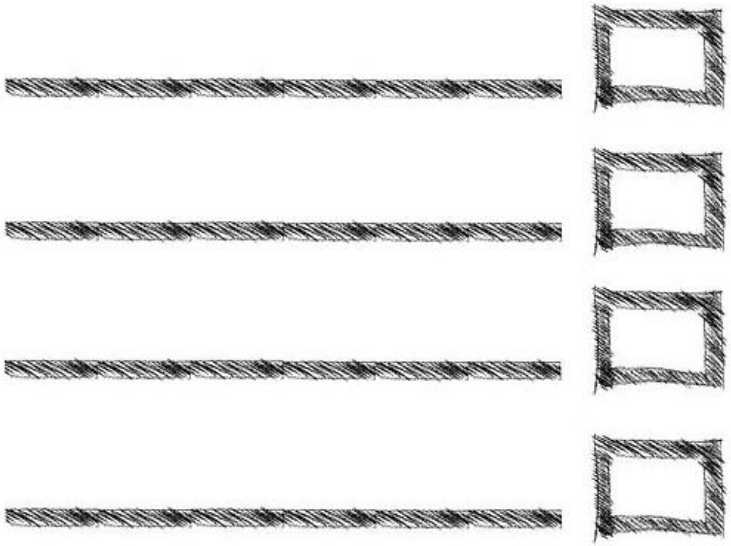


Image by Gerd Altmann from Pixabay

ADT `List` - Main operations

Method and Description
Add(T t) : void Adds the specified element to the end of this list
Insert(int index, T element) Inserts the specified element at the specified position in this list
AddRange(IEnumerable<T> c) Adds the elements of the specified collection to the end of this list
Item(int index) Gets or sets the element at the specified index
Contains(T o) Determines whether an element is in this list
Count Gets the number of elements contained in this list
RemoveAt(int index) Removes the element at the specified index of this list
Remove(T o) Removes the first occurrence of a specific object from this list
Clear() : void Removes elements from this list

https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1

Outline

- A review of Arrays
- What are Abstract Data Types (ADT)?
- List ADT
- **Using List ADT**
  - **Removing duplicate elements**
  - **Players-Team solution using Lists**
- Implementing List ADT

Quiz

What is the output of the following method?

```
static void Main() {
   List<string> myList = new List<string>();
   myList.Add("FOPCS");
   myList.Add("OOPCS");
   myList.Insert(2, "MVC.NET");
   if (myList.Contains("OOPCS"))
      myList.Insert(3, "Design");
   myList.Insert(1, "Data Structures");
   myList[3] = "Java";
   myList.RemoveAt(4);
   myList[1] = "Android";

   foreach (string module in myList)
      Console.WriteLine(module);
}
```

Do we care know how operation *Add()* is implemented?

Quiz

Given a **list of integers**

Write a static method that returns another list with **non-duplicate elements**, keeping only the **first occurrence**

Input: { 1, 5, 2, 4, **4**, 3, **5**, 7, 9, **2** }

Output: { 1, 5, 2, 4, 3, 7, 9 }

Players-Team Solution with List

Keep a **list of players** to add/remove players

```
class Team {
   public string TeamName { set; get; }
   private List<Player> players;

   public Team(string teamName) {
      players = new List<Player>();
      TeamName = teamName;
   }

   public void AddPlayer(Player player) {
      players.Add(player);
   }

   public void RemovePlayer(Player player) {
      players.Remove(player);
   }
```

Do we need to keep a separate variable ***numPlayers***?

Players-Team Solution with List

A list can be easily **traversed** with ***for*** or ***foreach*** loop

```
    public void Print() {
        Console.WriteLine(TeamName);
        for (int i = 0; i < players.Count; i++)
        {
            // Don't need to check against null
            Console.WriteLine(players[i]);
        }
    }
}
```

Next

How is the ADT List implemented?

Image by Wokandapix from Pixabay

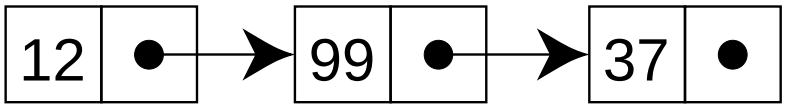
Outline

- A review of Arrays
- What are Abstract Data Types (ADT)?
- List ADT
- Using List ADT
- **Implementing List ADT**
  - **What are data structures?**
  - Implementing Lists using Arrays
  - Implementing Lists using Linked Lists

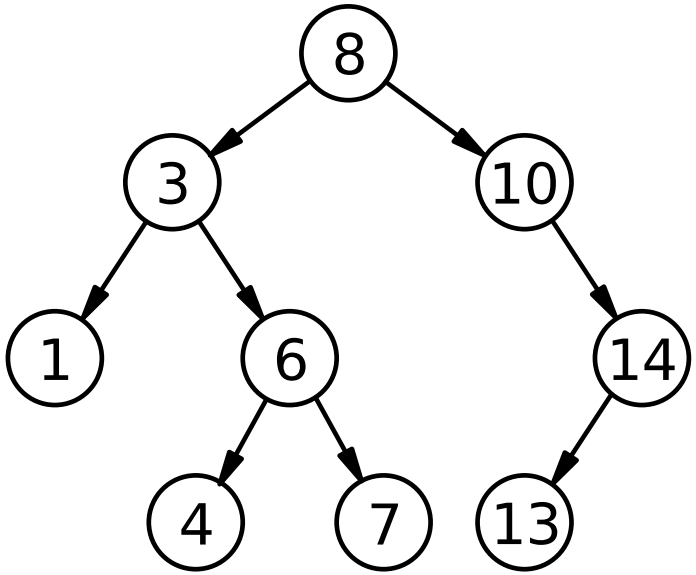
Data Structures

A data structure is a particular **scheme** of **organizing related data items**

Arrays



Linked List

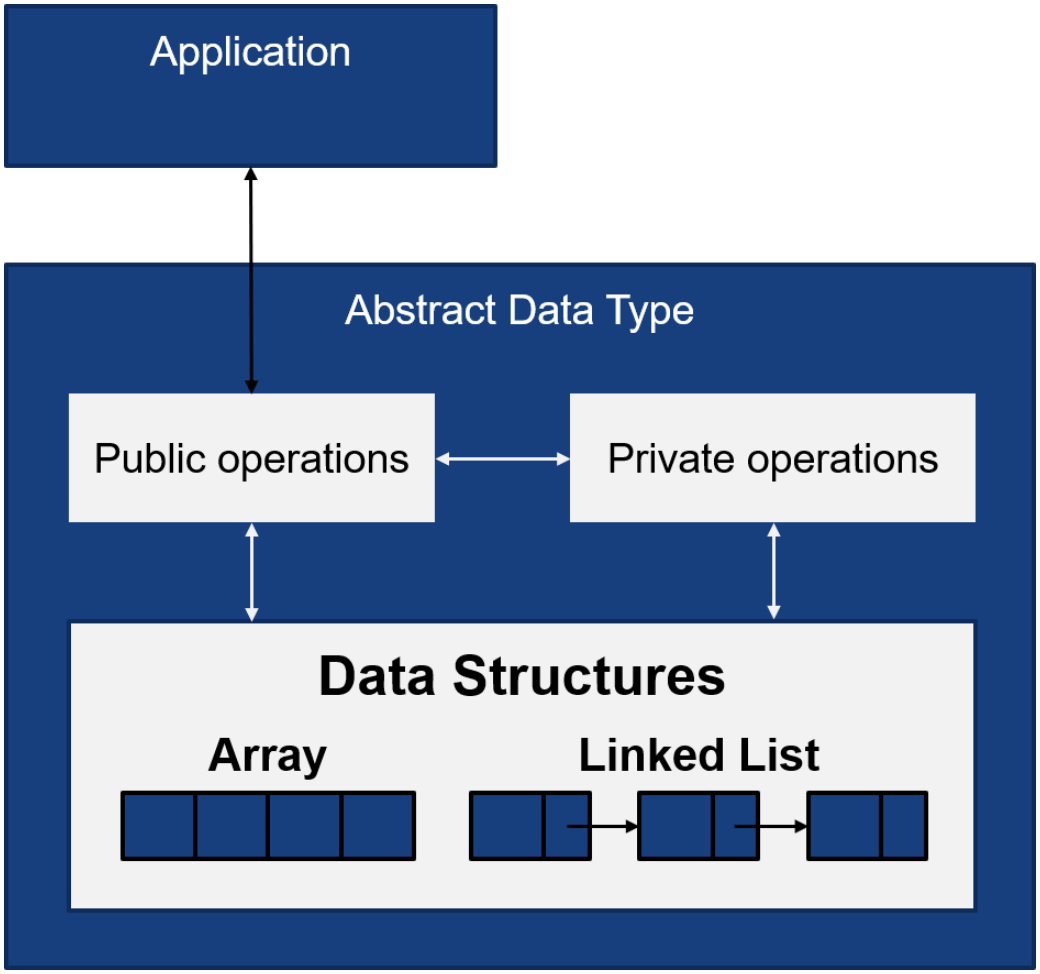


Binary Search Tree

Data Structures

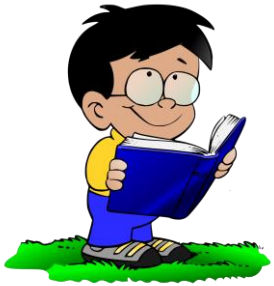
Data structures are **used to implement ADTs**

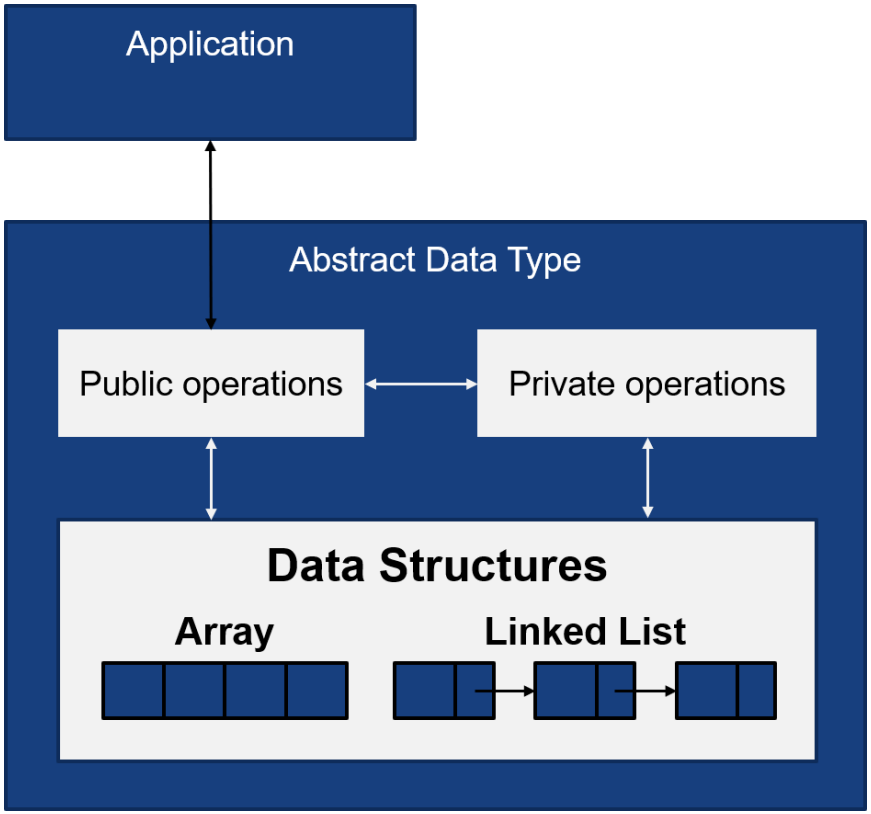
An ADT can be **implemented** in **different ways**, using **different data structures**



Question

What does it mean by "implementing an ADT"?





Implementing Lists using Arrays

- To make it simple, **let's implement a List of *`string`***
- For ***`int`***, ***`double`***, ***`Players`***..., just **replace *`string`*** by the respective data type
- Later in the course, we'll study ***Generics***, which allowed data types to be parameterized

Outline

- A review of Arrays
- What are Abstract Data Types (ADT)?
- List ADT
- Using List ADT
- **Implementing List ADT**
  - What are data structures?
  - **Implementing Lists using Arrays (Self Study)**
  - Implementing Lists using Linked Lists

Implementing Lists using Arrays

Self study

We keep **an array** to **store** the **list of strings**, and **another variable** to **store** the **number** of **elements**

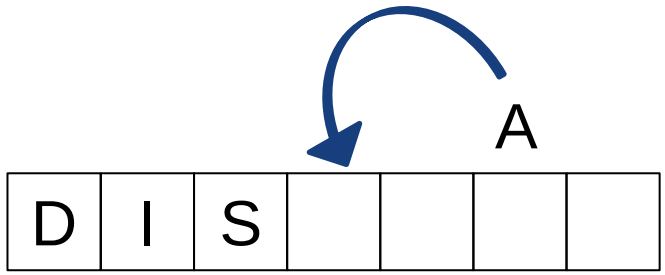
```
class AList
{
  private const int DEFAULT_CAPACITY = 10;
  private string[] arr;
  private int numElements;

  public AList()
  {
    arr = new string[DEFAULT_CAPACITY];
    numElements = 0;
  }
  // ...
}
```

Adding elements

Self study

After adding a new entry, **array capacity** needs to be **ensured**. In other words, use a **bigger array** if needed

```
public void Add(string newElement) {
  arr[numElements] = newElement;
  numElements++;

  EnsureCapacity();
}
private void EnsureCapacity() {
  int capacity = arr.Length - 1;
  if (numElements >= capacity) {
    // Replace with a new bigger array
    int newCapacity = capacity * 2;
    string[] newArr = new string[newCapacity];
    arr.CopyTo(newArr, 0);
    arr = newArr;
  }
}
```

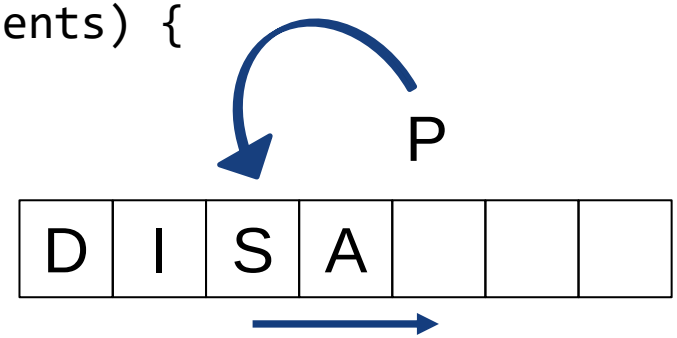


Inserting elements

Self study

When inserting an entry to a **specified position**, some entries needs to be **shifted** to the **right**

```
public void Insert(int index, string newElement) {
  // Allow inserting to the end
  if (index >= 0 && index <= numElements) {
    if (index < numElements)
      MakeRoom(index);
    arr[index] = newElement;
    numElements++;
    EnsureCapacity();
  } // else Invalid index
}

// Shift  entries toward the end of the array
private void MakeRoom(int index) {
  for (int i = numElements; i > index; i--)
    arr[i] = arr[i - 1];
}
```



Quiz – Implement method

Self study

Removes the element at the specified index from the list

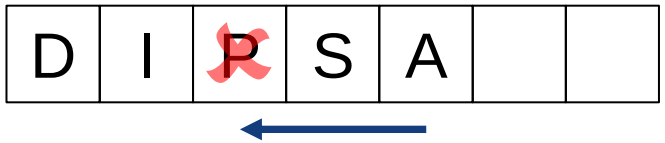
```
public void RemoveAt(int index)
```

Quiz Solution

Self study

When **removing** an entry from a specified position, **some elements** needs to be **shifted** to the **left**

```
public void RemoveAt(int index) {
  if (index >= 0 && index <= numElements - 1) {
    if (index < numElements - 1)
      RemoveGap(index);
    numElements--;
  } // else Invalid index
}

private void RemoveGap(int index) {
  for (int i = index; i < numElements - 1; i++) {
    arr[i] = arr[i + 1];
  }
}
```



Implementing Lists with Arrays

Self study

Study by yourself the implementation of:

- Method *`Replace(int index, string newElement)`*
- Method *`Contains(string element)`*
- Method *`GetAt(int index)`*

Using our AList

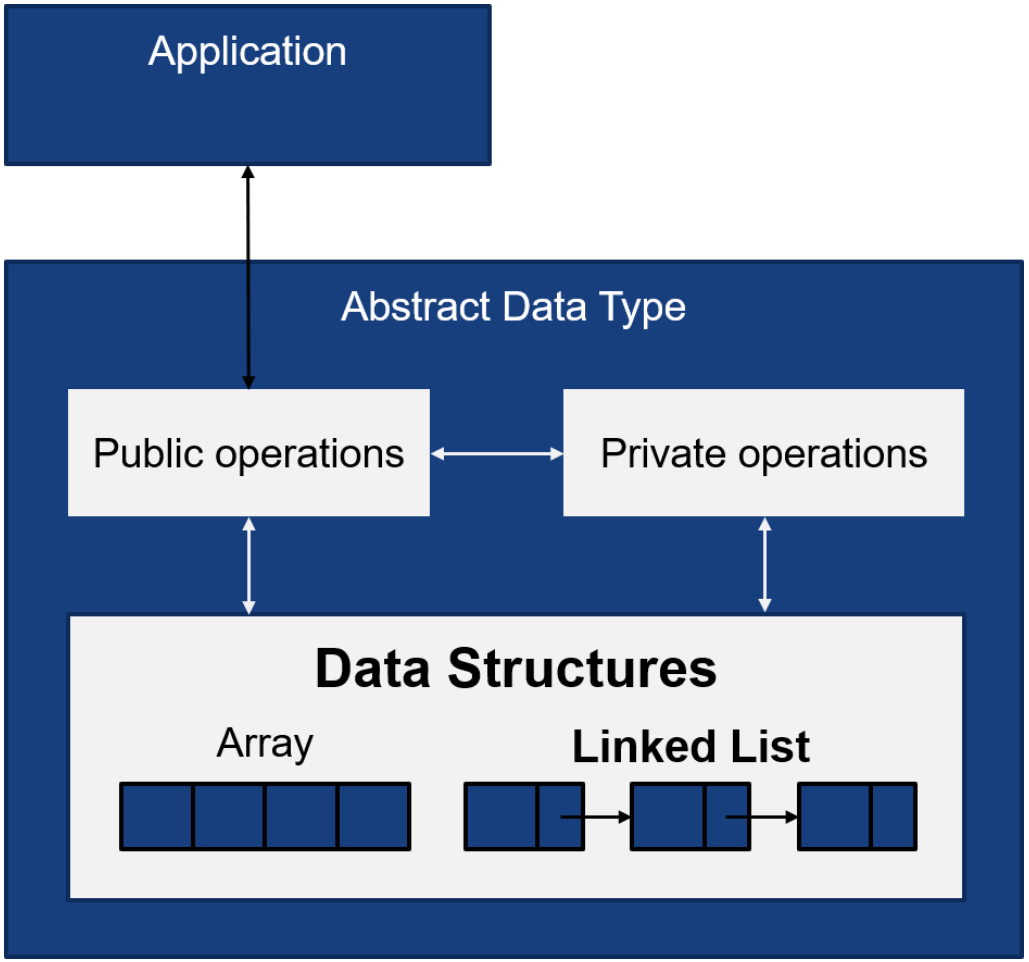
Self study

```
public static void Main() {
  AList myList = new AList();
  myList.Add("FOPCS");
  myList.Add("OOPCS");
  myList.Insert(2, "MVC.NET");
  if (myList.Contains("OOPCS"))
    myList.Insert(3, "Design");
  myList.Insert(1, "Data Structures");
  myList.Replace(3, "Java");
  myList.RemoveAt(4);
  myList.Replace(1, "Android");

  for (int i = 0; i < myList.Count(); i++)
    Console.WriteLine(myList.GetAt(i));
}
```

Does this slide look **familiar** ☺?

Next

Say hi to Linked List, an important **data structure** which can be used to **implement Lists** and **many other ADTs**

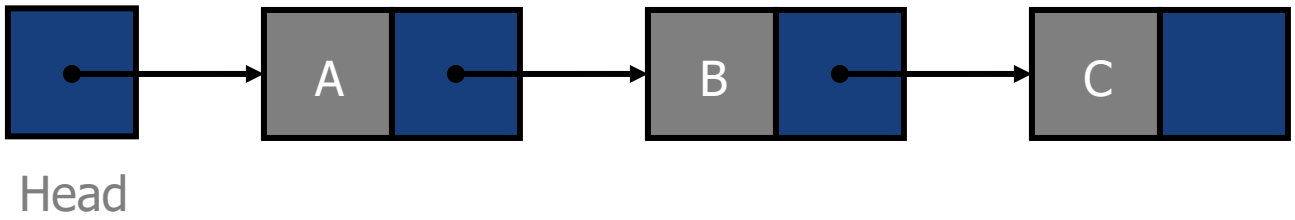


Outline

- A review of Arrays
- What are Abstract Data Types (ADT)?
- List ADT
- Using List ADT
- **Implementing List ADT**
  - What are data structures?
  - Implementing Lists using Arrays
  - **Linked List Data Structure**
  - Implementing Lists using Linked Lists

Linked List

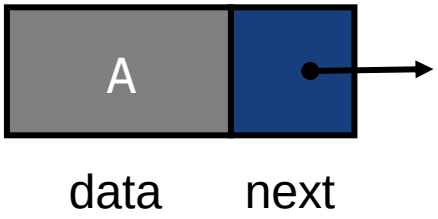
A linked list is a **data structure** that represents a **sequence** of **connected nodes**



Linked List Node

Each node contains at least **a piece of data** (of some type) and a **pointer to the next node** in the list

```
class Node
{
    public Node(string data)
    {
        Data = data;
        Next = null;
    }

    public string Data { set; get; }
    public Node Next { set; get; }
}
```



How can we implement ADT List using a Linked List?

Outline

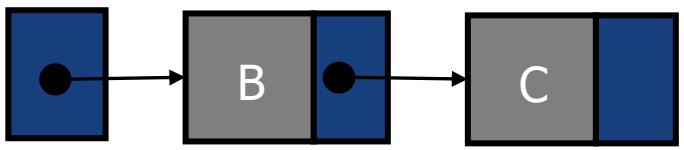
- A review of Arrays
- What are Abstract Data Types (ADT)?
- List ADT
- Using List ADT
- **Implementing List ADT**
  - What are data structures?
  - Implementing Lists using Arrays
  - Linked List Data Structure
  - **Implementing Lists using Linked Lists**

Implementing Lists with Linked List

Keep a **reference** to the **Head node** of the linked list, and the **number** of **elements**

```
class LList {
  public Node Head { set; get; }
  private int numElements;

  public LList() {
    Head = null;
    numElements = 0;
  }
  ...
}
```

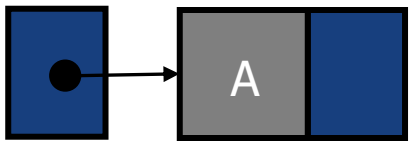
At the beginning, Head does NOT reference any node



Adding Elements

When adding to an **empty** Linked List, **Head** will reference the **new node**

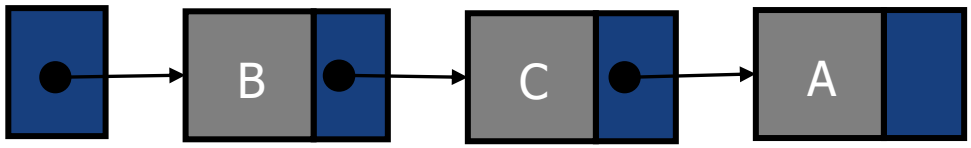
```
public void Add(string newElement) {
  Node newNode = new Node(newElement);
  if (numElements == 0) {
    Head = newNode;
  }
  else {
    Node lastNode = GetNodeAt(numElements - 1);
    lastNode.Next = newNode;
  }
  numElements++;
}
```



Adding Elements

When adding to a n**on-empty** Linked List, the **current last node** will **reference** the **new node**

```
public void Add(string newElement) {
  Node newNode = new Node(newElement);
  if (numElements == 0) {
    Head = newNode;
  }
  else {
    Node lastNode =
        GetNodeAt(numElements - 1);
    lastNode.Next = newNode;
  }
  numElements++;
}
```



Quiz – Implement method

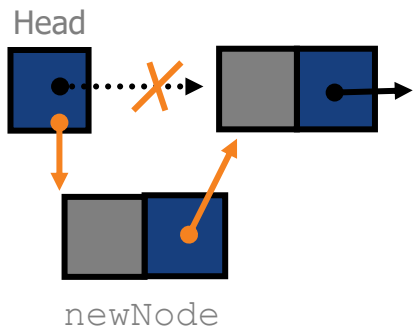
Implement a method to **retrieve** the **node at a given index**

```
private Node GetNodeAt(int index)
```

Inserting to a certain index

When inserted as the **first index**, the **new node** will become the **new Head**

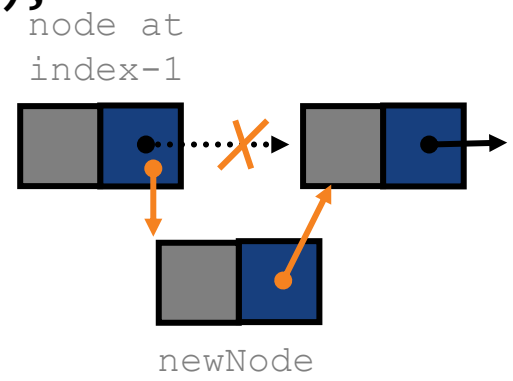
```
public void Insert(int index, string newElement) {
  if (index >= 0 && index <= numElements) {
    Node newNode = new Node(newElement);

    if (index == 0) {
      newNode.Next = Head;
      Head = newNode;
    }
    else {
      Node nodeBefore = GetNodeAt(index - 1);
      Node nodeAfter = nodeBefore.Next;

      nodeBefore.Next = newNode;
      newNode.Next = nodeAfter;
    }
    numElements++;
  } // else Invalid index
}
```



Inserting to a certain index

When inserted in the **middle** or at the **end** of the list, **links** for the **nodes before** and **after** will be **updated**

```
public void Insert(int index, string newElement) {
  if (index >= 0 && index <= numElements) {
    Node newNode = new Node(newElement);

    if (index == 0) {
      newNode.Next = Head;
      Head = newNode;
    }
    else {
      Node nodeBefore = GetNodeAt(index - 1);
      Node nodeAfter = nodeBefore.Next;

      nodeBefore.Next = newNode;
      newNode.Next = nodeAfter;
    }
    numElements++;
  } // else Invalid index
}
```

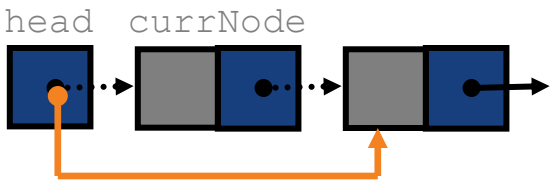


Removing from a given index

Self study

When removing the **first node**, **Head** updates to **reference** the **second node**

```
public void RemoveAt(int index) {
  if (index >= 0 && index <= numElements - 1) {
    if (index == 0) {
      Head = Head.Next;
    }
    else {
      Node nodeBefore = GetNodeAt(index - 1);
      Node nodeToRemove = nodeBefore.Next;
      Node nodeAfter = nodeToRemove.Next;

      nodeBefore.Next = nodeAfter;
    }
    numElements--;
  }
  // else // Incorrect index
}
```

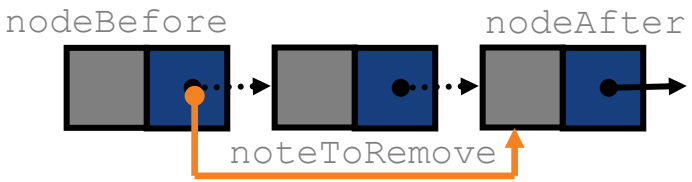


Removing from a given index

Self study

When removing a **node other** than the **first one,** the **previous node** updates to **reference after node**

```
public void RemoveAt(int index) {
  if (index >= 0 && index <= numElements - 1) {
    if (index == 0) {
      Head = Head.Next;
    }
    else {
      Node nodeBefore = GetNodeAt(index - 1);
      Node nodeToRemove = nodeBefore.Next;
      Node nodeAfter = nodeToRemove.Next;

      nodeBefore.Next = nodeAfter;
    }
    numElements--;
  }
  // else // Incorrect index
}
```



Implementing Lists with Arrays

Self study

Study by yourself the implementation of:

- Method *Replace(int pos, string newData)*
- Method *Contains(string entry)*
- Method *Count()*

What have we done so far?

Using Linked List, we have implemented a simplified version of List ADT with the following methods

```
class LList {
    public void Add(string newElement);
    public void Insert(int index, string newElement);
    public string GetAt(int index);
    public bool Contains(string element);
    public int Count();
    public void Replace(int index, string newElement);
    public void RemoveAt(int index);

    private Node GetNodeAt(int index);
}
```

Why is method *GetNodeAt()* private?

Using our LList

```
public static void Main() {
  LList myList = new LList();
  myList.Add("FOPCS");
  myList.Add("OOPCS");
  myList.Insert(2, "MVC.NET");
  if (myList.Contains("OOPCS"))
    myList.Insert(3, "Design");
  myList.Insert(1, "Data Structures");
  myList.Replace(3, "Java");
  myList.RemoveAt(4);
  myList.Replace(1, "Android");

  for (int i = 0; i < myList.Count(); i++)
    Console.WriteLine(myList.GetAt(i));
}
```

Does this slide look **familiar (again)** ☺?

Question

Image by Robin Higgins from Pixabay

So, we have **two implementations** for **the same** ADT List. Can we just **ignore Linked List** and **always use Array List** in our coding?

Readings

- Data structures and abstractions with Java, 4ed – Chapter 12, Lists, *Frank M.Carrano and Timothy M. Henry*
- Data structures and abstractions with Java, 4ed – Chapter 13, A List implementation that uses an Array, *Frank M.Carrano and Timothy M. Henry*
- Data structures and abstractions with Java, 4ed – Chapter 14, A List implementation that uses Links Data, *Frank M.Carrano and Timothy M. Henry*