Linear Data Structures

A data structure is said to be *linear* if its elements form a sequence or a linear list.

Examples:

- Arrays
- Linked Lists
- Stacks, Queues

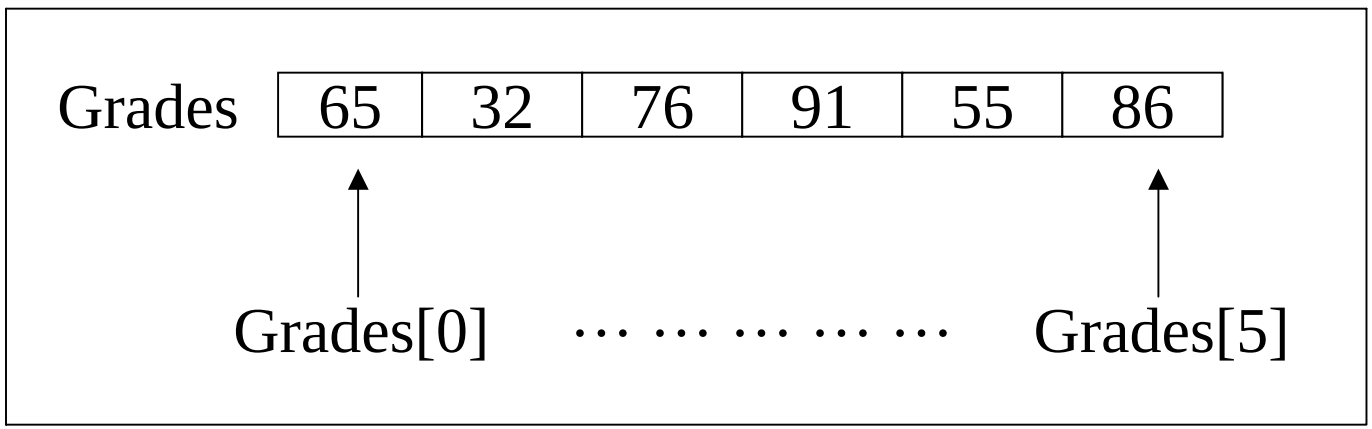
Operations on linear structures

- *Traversal*: Travel through the data structure
- *Search*: Traversal through the data structure for a given element
- *Insertion*: Adding new elements to the data structure
- *Deletion*: Removing an element from the data structure
- *Sorting*: Arranging the elements in some type of order
- *Merging*: Combining two similar data structures into one

Arrays

- Definition: A consecutive group of memory locations that all have the same name and of identical type

- Visual Example: grades for 6 students:



- Note that all array index references in Java start at 0

Array Terminology & Declaration

- Array Declaration

type ArrayName[ ] = ***new*** *type*[<array size>];

- Example:
```
float Grades[] = new float[7];
```

- `Grades` is an array of type `float` with size 7.

- `Grades[0], Grades[1], …, Grades[6]` are the **elements** of the Grades; each is of type `float`.

- 0,1,2,…,6 are the **indices** of the array. Also called **subscripts**. (Note that the indices start at 0 and NOT 1)

- During array declaration we may also put the brackets before the variable name:
  i.e. `float [ ]Grades = new float[7];`

Initialising Arrays

- Arrays may be initialised in the following ways:

  - int n[] = { 2, 4, 6, 8, 10 };
    creates and initialises a five element array with specified values

  - int n[] = new int[10];
    creates and initialises a 10 element array of zeros.

- Note : if data type is a non primitive type then above expression would create and initialise a 10 element array of nulls

Some Things To Note

- You Cannot assign data to arrays like such :

  list = { 1, 2, 3, 4, 5}; Wrong!

- Array elements are indexed between zero and the size of the array minus one

- Arrays can have any type

- You can check the size of your array by calling the ***length*** member variable.

e.g.

```
int anArray[] = new int[5];
System.out.println(anArray.length);
```

The above example will print out 5 to the terminal

Array Parameters

- You can pass arrays into functions as part of the function parameter like any other variable.

- e.g.

```
int results = new int[20];
        :
printResults(results);
```

- The function prototype for “printResults” was defined as such :

```
void printResults(int SomeArray[]);
```

- Note :
  - Arrays in Java are treated like objects thus all arrays are passed in by reference.

  - We don’t need to pass in the array size since we can get this from the length method.

Traversing linear arrays

- Usual way to traverse a linear 1-d array is to use a loop.
- e.g. Getting the overall average grade

```
Pseudo-Code : Get_Average(Array[])
        Begin
            index = 0;
            sum = 0;
            for index = 0  to index = ArraySize -1
            {
                  sum = sum + Array[index];
            }
            average = sum / ArraySize;
        End
```

Sample Java Code for example

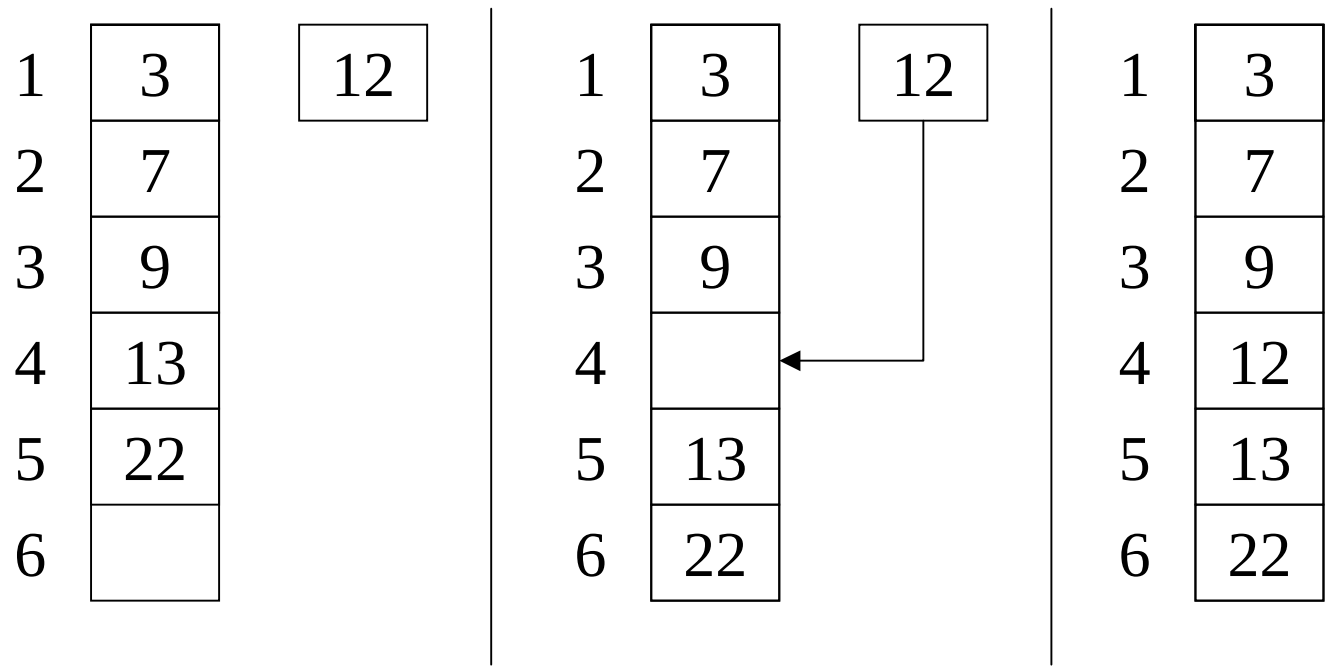
```
/*
 * function which calculates and returns the average overall grade
 */
float Get_Average(float Grades[]);
{
   float sum = 0;              // initialise the sum variable

     // We use a for loop for traversal here because
     // it's the handiest loop for what we want done
   for(int index = 0; index < Grades.length; index++)
   {
         sum += Grades[index];
   }

      // Return the average
    return sum/Grades.length;
}
```

Inserting elements

Adding an element to an array/list at an arbitrary position without overwriting the previous values requires that you move all elements "below" that position:



Algorithm

Algorithm: Insert(List[], position, element, ArraySize)

1. Start at the top element of the array.
2. Traverse the array backwards so as not to overwrite any previous data.
3. Replace the current element that we are on with the element before it.
4. Stop once we have reached our insertion position in the array.
5. Insert our data into that position

Sample Pseudo-Code

```
Pseudo-Code: Insert( List[], position, element)

        Begin
            for index = ArraySize-1 to index = position+1
            {
                    List[index] = List[index-1];
             }

           List[position] = element;
        End
```

Sample Java Code

```
// function definition for insert
void insert(int list[], int pos, int elem)
{
     // we are assuming that ‘pos’ is the array position
     // with the lower bound starting at zero and not one
   for(int i = list.length-1; i > pos ; i--)
        list[i] = list[i-1];

   list[pos] = elem;            // insert the element
}
```

Multidimensional Arrays

- Arrays can be more than one dimensional.
- Used to represent tables of data, etc.
- Declaration of 2-d array :

```
int grid[][] = new int[5][6];
```

- This declaration is interpreted as an array consisting of 5 rows and 6 columns
- Another way of declaring a 2-d array :

```
int grid[][] = new int[2][];
grid[0] = new int[2];
grid[1] = new int[2];
```

- Same as declaring a 2x2 array except we are doing it one row at a time here

- e.g. for 3-d array :

```
int space = new int[100][100][100];
```

Initialising Multidimensional Arrays

- e.g.

```
int array1[][] = {{1,2,3}, {4,5,6}};
int array2[][] = new int[3][2];
int array3[][] = {{1,2}, {4}};
```

- If we were to print out the values of these arrays by row we would get :

array1	array2	array3
1 2 3	0 0 0	1 2
4 5 6	0 0	4

- **Note :** If there are not initialisers for a given row, primitive types are initialised to zero and non primitive types are initialised to null.

Multidimensional Arrays as Parameters

- e.g.

```
void warp(int space[][][]);
```

- Same as declaring a one dimensional array as parameters except that we must remember to include extra square brackets for each additional dimension.