**Data Structures - Lecture Notes**

**Array Lists**

**1. The List ADT:**

A list is an ordered set of items. Ordered means that each element has a certain position. This is different than a sorted list in which the values themselves are ordered. The list allows for items to be added at any position and to be deleted from any position.

The list of functions associated with the list ADT are:

* Length: which returns the number of items in the list.
* At(i): returns the value stored at position i.
* InsertAt(i): adds an item to the list in position i.
* DeleteAt(i): removes the item at position i.
* Append: adds an item to the end of the list.
* Clear: deletes all values leaving an empty list.

**Implementations:**

There are two different possible implementations to the list ADT:

1. Linked Lists: Linked lists consist of a set of nodes each holding a value and a pointer to the next node.
2. Array List: It is simply a dynamic array whose size can change according to need.

**2. Array Lists:**

This is the second implementation of the list ADT which uses a dynamic array to store data. You can change the size of a dynamic array in runtime as opposed to static arrays.

In practice a dynamic array is a pointer to an array. The pointer holds the address of the first element in the array. When you need to change the size of the array you will need to create a new array with the new size and change the pointer to point at the newly created array.

The spaces in the dynamic array are consecutive spaces in memory. This allow you to increment the pointer and it will point to the next element in the array.

**Implementation:**

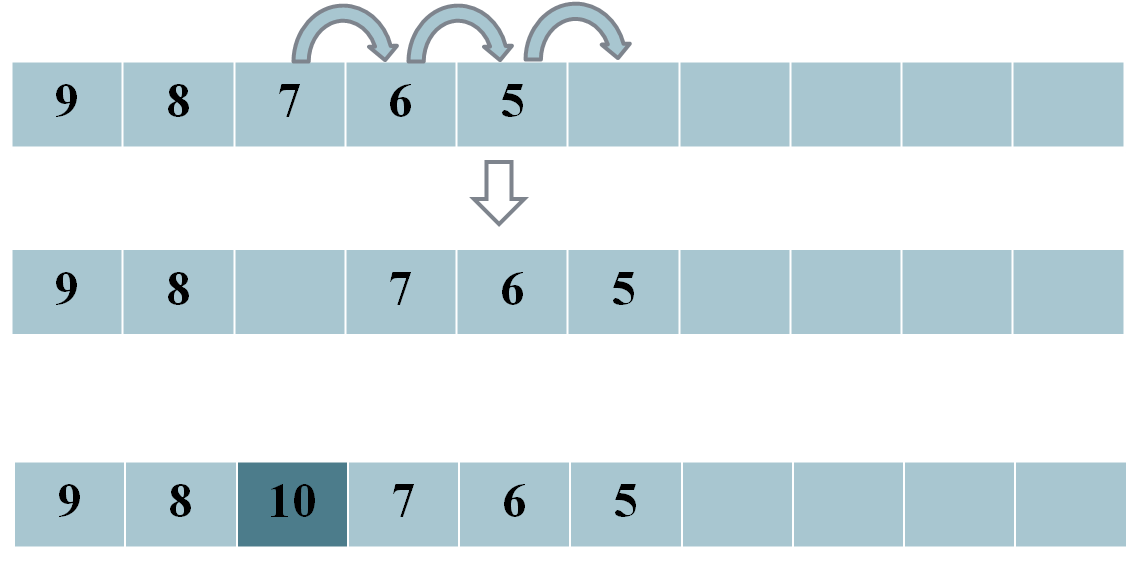
**Class Declaration:**

We need to maintain the pointer to the array and two count parameters, size indicates the current capacity of the array (including the empty spaces) and elems indicates the actual number of values saved in the array.

|  |
| --- |
| template <class T>  class ArrList  {  public:  T\* arr;  int size, elems;  public:  ArrList(int initSize);  int Length();  void Append(T val);  void Expand();  T At(int pos);  void insertAt(int pos, T val);  void deleteAt(int pos);  ~ArrList(void);  }; |

***InsertAt:***

The function inserts an element at a specific position. You will need to shift all the elements starting from the last element until you reach the required position so that you can leave an empty space for the new value. The process is explained in figure 1.



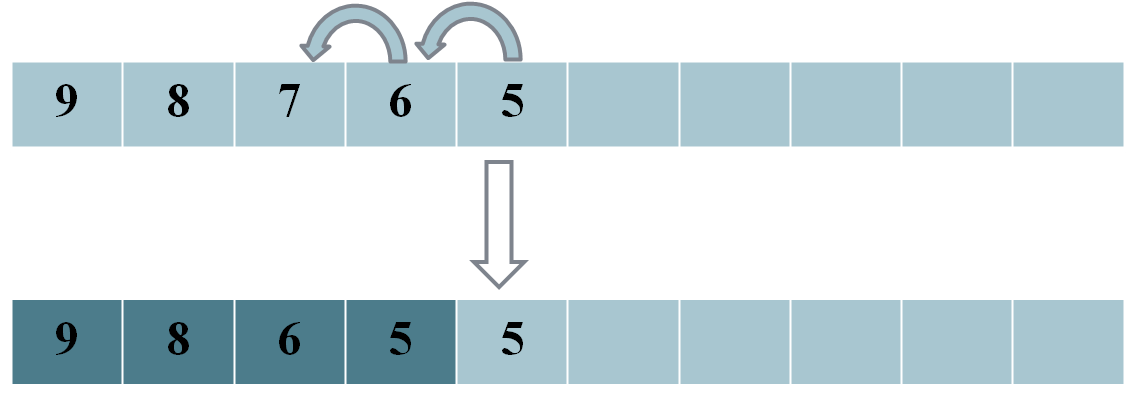
**Figure 1. The InsertAt Operation**

At the start the function needs to check if the position given by the user is a valid position, that is it should not exceed the number of elements. Given the process described above the loop counter to shift the elements should start at the end of the array and work backwards to the required position. Once the required position is empty the new value can be placed in the array.

|  |
| --- |
| template <class T>  void ArrList<T>::insertAt(int pos, T val)  {  assert(pos<elems);  if(elems==size)  Expand();  //shift elements to the right  for(int i=elems;i>pos; i--)  arr[i]=arr[i-1];  //insert value at required position  arr[pos]=val;  elems++;  } |

***DeleteAt:***

The function deletes an element at a specific position. You will need to shift all the elements starting from the required position and then move forward to replace the deleted value. It leaves the last element but since we update the number of elements each time we add/ remove an item, the number of elements will point to the actual number of element so the last element will not be included in the processing. The process is explained in Figure 2.



**Figure 2. The DeleteAt Operation**

|  |
| --- |
| template <class T>  void ArrList<T>::deleteAt(int pos)  {  assert(pos<elems);  //shift elements to the left  for(int i=pos;i<elems-1;i++)  arr[i]=arr[i+1];  elems--;  } |