Data Abstraction and Encapsulation

By Omieno K. Kelvin

Computer science department-MMUST

- Data Types
  - A *data type* is a collection of *objects* and a set of *operations* that act on those objects.
  - Natural data types
    - Books
    - Students
    - Beers
  - Simple computer data types (often predefined in a programming language)
    - int
    - float
    - char
  - Aggregate computer data types (often user defined)
    - Arrays, e.g.,
    - ```
      const int MAX_STRING_LENGTH = 80;
      typedef char string[MAX_STRING_LENGTH + 1];
      ```
    - structs, e.g.,
    - ```
      typedef struct {
          string StudentName;
          int NumberOfBooks;
          char Grade;
      } student_record;
      ```

- Abstract Data Types
  - *Data abstraction* separates the *specification* of a data type from its *implementation*.
  - An *abstract data type* is a specification of a collection of objects and a set of operations that act on those objects.
- Some ADTs
  - The Natural Number ADT
  - Natural data types
  - Simple computer data types
  - Aggregate computer data types
- Benefits of Using ADTs
  - Modularity: each data type can be considered independently.
  - Testing: data type implementations can be tested separately to check that they meet the specifications.
  - Division of labour: separate implementation of data types according to agreed specifications.
  - Reusability: implemented data types can be reused.

    - Change of implementation: the implementation can be changed with no effect to users.
- Data Encapsulation
    - *Data encapsulation* conceals the implementation of a data type from the user.
    - Benefits
        - Change of implementation: the implementation can be changed with no effect to users.
        - Prevents corruption: users are not able to manipulate the data directly, hence incorrectly.
    - Data encapsulation is a natural technique when implementing ADTs, due to the

Exam Style Questions

- Define {data type,data abstraction,data encapsulation,abstract data type}.
- Give an example of an abstract data type.
- List five benefits of using ADTs, giving a short explanation of each.
- List two benefits of using data encapsulation, giving a short explanation of each.

The Array as an ADT

- The Array as an ADT
    - An array object is a set of pairs, <index,value>, such that each index is unique and each index that is defined has a value associated with it (a mapping from indices to values).
    - Operations include setting and retrieving a value for a given index.
    - An array ADT
- Representation of Arrays
    - Space requirements
        - Product of dimensions
    - Row and column major orders
        - Row major changes column (in general, last index) fastest
        - Column major changes row (in general, first index) fastest
    - Address calculation
        - Assume one address per element, scale later if necessary. Assume base address alpha.
        - In 2D
            - Array A[NumberOfRows][NumberOfColumns]
            - Offset for preceding rows is Row * NumberOfColumns
            - Offset in row is Column
            - Address is alpha + Row * NumberOfColumns + Column
        - In 3D
            - Array A[NumberOfSlices][NumberOfRows][NumberOfColumns]
            - Offset for preceding slices is Slice * NumberOfRows * NumberOfColumns
            - Offset for preceding rows is Row * NumberOfColumns

        - Offset in row is Column
        - Address is alpha + Slice * NumberOfRows * NumberOfColumns + Row * NumberOfColumns + Column
- In ND
    - Array A[u1][u2]...[uN]
    - Offset for dimensions preceding K is iK * uK+1 * uK+2 * ... * uN
    - Offset is (see HSM p.102)

An array is a number of data items of the same type arranged contiguously in memory. The array is the most commonly used data storage structure; it's built into most programming languages. Because they are so well-known, arrays offer a convenient jumping-off place for introducing data structures and for seeing how object-oriented programming and data structures relate to each other

An Array Example

Let's look at some sample programs that show how an array can be used. In case you're making the transition to OOP, we'll start with an old-fashioned procedural version, and then show the equivalent object-oriented approach. Listing 2.1 shows the old-fashioned version, called array.cpp.

```
//array.cpp
//demonstrates arrays
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
int main()
{
int arr[100]; //array
int nElems = 0; //number of items
int j; //loop counter
int searchKey; //key of item to search for
//-----------------------------------------------------------
arr[0] = 77; //insert 10 items
arr[1] = 99;
arr[2] = 44;
arr[3] = 55;
arr[4] = 22;
arr[5] = 88;
arr[6] = 11;
arr[7] = 00;
arr[8] = 66;
```

```
arr[9] = 33;
nElems = 10; //now 10 items in array
//-------------------------------------------------------------
for(j=0; j<nElems; j++) //display items
cout << arr[j] << " ";
cout << endl;
//-------------------------------------------------------------
searchKey = 66; //find item with key 66
for(j=0; j<nElems; j++) //for each element,
if(arr[j] == searchKey) //found item?
break; //yes, exit before end
if(j == nElems) //at the end?
cout << "Can't find " << searchKey << endl; //yes
else
cout << "Found " << searchKey << endl; //no
//-------------------------------------------------------------
searchKey = 55; //delete item with key 55
cout << "Deleting " << searchKey << endl;
for(j=0; j<nElems; j++) //look for it
if(arr[j] == searchKey)
break;
for(int k=j; k<nElems; k++) //move higher ones down
arr[k] = arr[k+1];
nElems--; //decrement size
//-------------------------------------------------------------
for(j=0; j<nElems; j++) //display items
cout << arr[j] << " ";
cout << endl;
return 0;
} //end main()
```

In this program, we create an array called arr, place 10 data items (kids' numbers) in it, search for the item with value 66 (the shortstop, Louisa), display all the items, remove the item with value 55 (Freddy, who had a dentist appointment), and then display the remaining nine items.

The output of the program looks like this:

77 99 44 55 22 88 11 0 66 33

Found 66

77 99 44 22 88 11 0 66 33

The data we're storing in this array is type int. We've chosen a basic type to simplify the coding. Generally the items stored in a data structure consist of several data members, so they are represented by objects rather than basic types. We'll see an example of this in the next hour.

Inserting a New Item

Inserting an item into the array is easy; we use the normal array syntax arr[0] = 77;

We also keep track of how many items we've inserted into the array with the nElems variable.

Searching for an Item

The searchKey variable holds the value we're looking for. To search for an item, step through the array, comparing searchKey with each element. If the loop variable j reaches the last occupied cell with no match being found, the value isn't in the array. Appropriate messages are displayed: Found 66 or Can't find 27.

Deleting an Item

Deletion begins with a search for the specified item. For simplicity we assume (perhaps rashly) that the item is present. When we find it, we move all the items with higher index values down one element to fill in the "hole" left by the deleted element, and we decrement nElems. In a real program, we would also take appropriate action if the item to be deleted could not be found.

Displaying the Array Contents

Displaying all the elements is straightforward: we step through the array, accessing each one with arr[j] and displaying it.

Summary

- In this tutorial, you've learned the following:
- Unordered arrays offer fast insertion but slow searching and deletion.
- Wrapping an array in a class protects the array from being inadvertently altered.
- A class interface comprises the member functions (and occasionally data members) that the class user can access.

- A class interface can be designed to make things simpler for the class user (although possibly harder for the class designer).

Exercises

- On average, how many items must be moved to insert a new item into an unsorted array with N items?
- 2. On average, how many items must be moved to delete an item from an unsorted array with N items?
- 3. On average, how many items must be examined to find a particular item in an unsorted array with N items?
- Implementing a one dimensional array ADT