Arrays

1E3
Topic 11

Objectives

- This topic should allow students to
  - Recognise when to use arrays in a program
  - Define and use arrays

- This topic is based on Chapter 9 of the textbook.
What is an array?

- An array is a collection of data items of the same type. E.g.
  - An array of 20 integers.
  - An array of 100 Student objects
  - An array of 5 first names

<table>
<thead>
<tr>
<th></th>
<th>Tom</th>
<th>Joe</th>
<th>Liz</th>
<th>Bob</th>
<th>Ann</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

An array is ordered

- There is an order to the slots in the array.
- Each slot is numbered.
  - In C++ slots are numbered from 0 upwards.
- We can put values of the appropriate type into each slot.
- We can get the values back out of each slot.
Why do we need arrays?

- Suppose we need to output how much each of a series of student grades differs from the average.

<table>
<thead>
<tr>
<th>Name</th>
<th>Grade</th>
<th>Diff from ave</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>55</td>
<td>+5</td>
</tr>
<tr>
<td>Mary</td>
<td>40</td>
<td>-10</td>
</tr>
<tr>
<td>Tom</td>
<td>60</td>
<td>+10</td>
</tr>
<tr>
<td>Ann</td>
<td>55</td>
<td>+5</td>
</tr>
<tr>
<td>Bill</td>
<td>40</td>
<td>-10</td>
</tr>
<tr>
<td>Eoin</td>
<td>50</td>
<td>0</td>
</tr>
</tbody>
</table>

- But we don’t know the average till we’ve seen all values.

Need for arrays

- If we know that there are only say 10 students, we could define 10 variables to hold the 10 grade objects.

```cpp
string name1, name2, name3, name4, ...;
double grade1, grade2, grade3, ...;
...
cin >> name1 >> grade1;
cin >> name2 >> grade2;
...
```
Need for arrays

- But this won’t work for large numbers of grades, or unknown numbers of grades.
- An array allows us to store lots of names and grades at one time.
  - We can then look at each when computing the average.
  - And then print each one out in the same order they were read in.

An array solves the problem

- An array of 6 name/grade objects

<table>
<thead>
<tr>
<th>Name</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>55</td>
</tr>
<tr>
<td>Mary</td>
<td>40</td>
</tr>
<tr>
<td>Tom</td>
<td>60</td>
</tr>
<tr>
<td>Ann</td>
<td>55</td>
</tr>
<tr>
<td>Bill</td>
<td>40</td>
</tr>
<tr>
<td>Eoin</td>
<td>50</td>
</tr>
</tbody>
</table>

- In this diagram, each array slot contains an object of a new data type SGrade which consists of a name and a grade.
- Think through the algorithm for the problem using this array.
Switch to a simpler problem

- Let’s simplify the problem for now.
  - Forget the names. Just read in a long list of grades
  - Output like this:

<table>
<thead>
<tr>
<th>Grade</th>
<th>Diff from ave</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>+5</td>
</tr>
<tr>
<td>40</td>
<td>-10</td>
</tr>
<tr>
<td>60</td>
<td>+10</td>
</tr>
<tr>
<td>55</td>
<td>+5</td>
</tr>
<tr>
<td>40</td>
<td>-10</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
</tr>
</tbody>
</table>

This problem can be solved with a simple array of doubles.

Declaring arrays

- double grades[6];
- declares an array called grades
- that can hold up to 6 values of type double.
- The slots in this array are numbered 0 to 5.

grades

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

6 is the declared size of this array.
Declaring arrays cont./

- When you declare an array its size must be known.
- You can’t use a size provided by the user.

```cpp
int arraysize;
cin >> arraysize;
int alist[arraysize]; //illegal
```

Accessing array elements

- The slots in the grades array are called `grades[0]`, `grades[1]`, ..., `grades[5]`
- In `grades[1]`, 1 is the subscript or index.
- `grades[1]` is called an indexed variable.
- An indexed variable can be used anywhere an ordinary variable (of the same type) can.
Accessing array elements

- `cin >> grades[1];`
  reads the next input value into the slot numbered 1 of the grades array.
- `x = grades[i];`
  gets the value out of the slot numbered by whatever value `i` currently has, and assigns it to `x`.
- Be careful -
  - `grades[i]` is referred to as the `i`th slot of `grades`,
  - but if `i` is 2, `grades[i]` refers to the third slot, the one numbered 2.

Array Index expressions

- The index can be any expression whose value is an integer between 0 and (arraysize - 1).
- `grades [i] = grades[i+2];`
  gives the `i`th slot the value of the `i+2`th slot
Use *for* loops for arrays

- *for* loops are perfect for handling all elements of an array.

  ```cpp
  for (i = 0; i < 6; i++)
  {
    cout << grades[i];
  }
  ```

- Always use the form

  ```cpp
  for (i = 0; i < arraysize; i++)
  ```

Processing a known number of grade values

- Later we will see how to handle collections of values whose size is unknown in advance.

- But first let’s look at the program to handle a known number of grades, e.g. 6

- See firstarray.cpp
Index Out of Range errors

- Serious errors will result from trying to access an array element beyond the end of the array.
- C++ does not check if the index value is within range.
- For `for (i = 1; i <= 6; i++)`
  `cout << grades[i];`
  will result in `cout << grades[7];`  
  **ERROR:**
  Possibly disastrous

Restrictions on array processing

- You can’t assign a whole array to another one.
  - You have to copy element by element
    `myarray = yourarray; //illegal`
- Instead
  `for (i=0; i<asize; i++)`
  `myarray[i] = yourarray[i];`
Restrictions cont./

- You can’t cin a whole array of elements
  - You have to read them in one at a time

  ```cpp
  cin >> myarray; // illegal
  ```

- Instead
  ```cpp
  for (i=0; i<asize; i++)
    cin >> myarray[i];
  ```

Common array processing patterns

- Initialise the elements of an array
- Read elements into an array
- Print all the elements of an array
- Sum the elements of an array
- Find the max of the elements of an array
- Write loops for all these.
Other notes

- Note the two uses of [ ] brackets.
  - `int myarray[10];`
    - 10 is the declared size of the array `myarray`
  - `x = myarray[6];`
    - 6 is the index or slot number being accessed.

Some array exercises
Reverse the input

- Write a program to read in 10 words and then write them out in reverse order.
  - How big an array will you need?
  - What type of data will the slots of the array hold?
  - Design the loop to read in the words.
  - Write the loop to write out the words starting at the end of your array.
- See reversesequence.cpp

Array of month names

- Write a program to map month numbers to month names, using an array.
  - Create an array of month names and fill it up.
    - Use slots numbered 1 to 12, and leave slot 0 blank.
    - Repeatedly ask user for a month number and return the month’s name by looking in the appropriate slot of the array.
  - This is a common use of arrays - to store a mapping.
Test the order of numbers

- Write a program to read in 5 numbers and to print out “In order” if each number is greater than or equal to its predecessor. Otherwise print “Not in order”.
- Decide on the type and size of array.
- When checking the ordering be careful not to access beyond the end of the array.
- Use a while loop instead of for loop to stop if and when you find a pair out of sequence.
- See ordertest.cpp

Next

- Next we’ll
  - write functions that handle arrays
  - see how to handle unknown numbers of array elements.
- Later we’ll
  - develop more interesting algorithms.
  - (maybe) use 2-dimensional arrays and multi-dimensional ones.