Session 7
Introduction to Arrays
Designed by Andrew Kwok-Fai Lui and Maria Poon

All rights reserved. No part of this document may be reproduced, transmitted, or stored in a retrieval system, in any form or by any means, without permission in writing by the author.

Copyright © by Andrew Kwok-Fai Lui and Maria Poon 2002, 2003

Note: To save space, the code for pausing the screen in Borland C++ Builder has been removed from all programs. You should add the `fflush(stdin)` and `getchar()` lines back in if you want to test the programs on Borland C++ Builder.

Achieve Target 1

We have focussed on program flow control using selection (if-else) and iteration (for-while) structures so far. We will look at a new feature in organizing data in your programs – arrays. Study the following program.

```
#include <stdio.h>

void main() {
    int anArray[1];
    printf("Enter an integer: ");
    scanf("%d", &anArray[0]);
    printf("The integer entered is %d\n", anArray[0]);
}
```

Output
Enter an integer: 23
The integer entered is 23

The above program reads in an integer and prints it out again – a pattern that you have seen a lot before. The key difference is the use of an array element to store the integer.

The program requires only one integer variable to store the integer. The line with a red circle declares an array of integer with the size of one. The size of an array indicates how many array elements there are in the array. Each integer array element can store an integer. So we have an array size one to satisfy the program requirement of one integer variable. The number within the square brackets ([1]) specifies the array size, and the name (identifier) of the array is `anArray`. Remember it.

The name `anArray` refers to the whole array of integer elements. Each array element is identified by a number (or index). The first element is numbered 0, the second is numbered 1, and so on. We use this index number to refer to an individual integer element, and we enclose the index number within a pair of square brackets. In the above program, `anArray[0]` represents the first element in the array `anArray`. In other words, `anArray[0]` can be used as an integer variable. Remember it.

Modify the above program so that instead of printing the entered number, we now want the square of the number be printed. The square of the number is the number multiplied by itself. Work on the program to make it happen.
Achieve Target 1 Cont

An array with just one array element (sized 1) is not more useful than an ordinary variable. The following program uses a float array with two array elements.

```c
#include <stdio.h>

void main() {
    float anArray[2];
    float sum;

    printf("Enter first number: ");
    scanf("%f", &anArray[0]);

    printf("Enter second number: ");
    scanf("%f", &anArray[1]);

    sum = anArray[0] + anArray[1];
    printf("The sum is %f\n", sum);
}
```

Output

Enter first number: 20
Enter second number: 12
The sum is 32.000000

The statement in green declares a float array with two array elements. Again the number within the square bracket ([2]) of the green statement indicates the size of the array, and float is the type of the array elements.

The two array elements are referred as anArray[0] and anArray[1]. The array index number always starts from 0. So the first element is numbered 0, and the second element is numbered 1, and so on. Remember it.

Modify the above program so that instead of using the variable sum to store the sum of anArray[0] and anArray[1], the new program uses the third array element. Pay attention to the following issues.

1. Declare the array size correctly – we now need three array elements instead of 2.
2. Refer to the third array element correctly – remember that the first element is indexed 0.

Work on the program to make it happen.

```
anArray

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
</table>

Each array element is same as a float variable
```
Achieve Target 1 Cont 2

The program below is the familiar BMI calculator program worked on previously in Perform Exercise.

```c
#include <stdio.h>

void main() {
    float height, weight, bmi;
    printf("Enter your Height(m): ");
    scanf("%f", &height);
    printf("Enter your Weight(kg): ");
    scanf("%f", &weight);
    bmi = weight / (height * height);
    printf("Your BMI = %f\n", bmi);
}
```

Rewrite the above program so that the program uses only a float array instead of using the three float variables. Work on the program to make it happen. Pay attention to the following issues.

1. Declare the array size correctly – how many array elements does the program need?
2. Designate each array element for holding the height, weight, and bmi, and follow the designation consistently.
Achieve Target 2

Study the following program.

```c
#include <stdio.h>

void main() { 
    float anArray[3];
    float sum;

    printf("Enter first number: "); 
    scanf("%f", &anArray[0]);

    printf("Enter second number: ");
    scanf("%f", &anArray[1]);

    printf("Enter third number: ");
    scanf("%f", &anArray[2]);

    sum = anArray[0] + anArray[1] + anArray[2];
    printf("The sum is %f\n", sum);
}
```

The program reads in three numbers, calculates their sum, and prints the sum out. The array size is 3, for each element is used to store one number. The sum is stored in a separate variable `sum`. This program and the previous ones used each array element as an ordinary variable.

The following program behaves exactly as the above but it uses arrays in a more typical manner – with a loop.

```c
#include <stdio.h>

void main() { 
    float anArray[3];
    float sum = 0;
    int i;

    printf("Enter first number: ");
    scanf("%f", &anArray[0]);

    printf("Enter second number: ");
    scanf("%f", &anArray[1]);

    printf("Enter third number: ");
    scanf("%f", &anArray[2]);

    for (i=0; i<3; i++)
        sum = sum + anArray[i];

    printf("The sum is %f\n", sum);
}
```

The `for` loop will execute three times, with the variable `i` changing from 0, 1, and to 2. Note that the variable `i` is used as the array index within the loop. So in the first execution of the loop, `anArray[0]` is referred; in the second executed of the loop, `anArray[1]` is referred, and so on. All the 3 array elements are in turn added to the variable `sum`. Therefore at the end of the loops, the variable `sum` will have the value the sum of the three array elements. Learn it.
Achieve Target 2 Cont

In a similar manner, the input part can also be replaced by a loop.

```c
#include <stdio.h>

void main() {
    float anArray[3];
    float sum = 0;
    int i;

    printf("Enter 3 numbers:\n");
    for (i=0; i<3; i++)
        scanf("%f", &anArray[i]);

    for (i=0; i<3; i++)
        sum = sum + anArray[i];

    printf("The sum is %f\n", sum);
}
```

Output

```
Enter 3 numbers:
2
3
4
The sum is 9.000000
```

The program has two for loops – one for input and one for calculation. Both for loops go through every array element one by one by **indexing**. **Learn** it.

The above program is again rewritten as the following equivalent program.

```c
#include <stdio.h>

void main() {
    float anArray[3];
    float sum = 0;
    int i;

    printf("Enter 3 numbers:\n");
    for (i=0; i<3; i++) {
        scanf("%f", &anArray[i]);
        sum = sum + anArray[i];
    }

    printf("The sum is %f\n", sum);
}
```

Only one for loop is needed for both input and summation. Do you remember this pattern? We have studied this type of programs in the Perform Exercises on Loops. The only new feature is the use of an array to store the input numbers, while previously the numbers are discarded after added to the sum. We have a rather easy-to-use method to store and process a set of numbers with arrays. **Remember** it.

Modify the above program so that 10 numbers are read in and the sum calculated. **Work** on the program to make it happen.
Achieve Target 3

Study the following program.

```c
#include <stdio.h>

void main() {

    int anArray[5];
    int i;

    printf("Enter 5 integers:\n");
    for (i=0; i<5; i++) {
        scanf("%d", &anArray[i]);
    }

    printf("The integers entered in reverse are:\n");
    for (i=4; i>=0; i--)
        printf("%d ", anArray[i]);
}
```

The program reads in 5 integers into an array of 5 elements, and then it prints them out in reverse order. The second `for` loop is written so that the index `i` begins from 4 and counting downwards to 0. The fifth array element (which is the last element in this array) is referred by the index of 4. The array elements are referred to from the last to the first. The integers entered are stored from the first element to the last, and the printing is in the order from the last to the first.

Modify the above program so that,

1. It prints the integers in the original order (in the order they are entered).
2. It prints the even number integers only, in the original order.
3. It prints the first, third, fifth, elements of the array (remember the first element is indexed 0), but in the reverse order.

Work on the program to make it happen.
Achieve Target 3 Cont

Study the following program.

```c
#include <stdio.h>

void main()
{
    char charArray[256];
    int length;
    int i;

    printf("Enter a line of text (press Enter at the end):
    ");
    for (i=0; i<256; i++)
    {
        scanf("%c", &charArray[i]);
        if (charArray[i] == \n) {
            length = i;
            break;
        }
    }

    printf("The text entered is:\n")
    for (i=0; i<length; i++)
    {
        printf("%c", charArray[i]);
    }
}
```

Output

Enter a line of text (press Enter at the end):
Array is a useful data structure
The text entered is:
Array is a useful data structure

The program illustrates the use of an array of char. The conversion character is now "%c" instead of "%d" or "%f". The scanf statement could be replaced by a getchar statement with no difference made. The following two statements are equivalent.

```c
scanf("%c", &charArray[i]);

charArray[i] = getchar();
```

Note that the first for loop is terminated immaturely if the entered character is "\n" (newline). It serves two purposes. The first purpose is that once the newline character is seen (indicating that the Enter key is pressed), the program should not attempt to read more characters. The second purpose is to record how many characters have been stored. This information could be referred from the index i, which points to the array elements holding the newline character. In other words, the index i indicates the length of the text stored in the array. The program assigns the index i to a variable length for record. Not the whole charArray has been stored with characters, so the variable length is the key to keep track of how many elements of charArray have been used. Remember it.

The second for loop is for printing the text. The variable length plays the crucial role here to control the execution of loops to match the length of the text.

Modify the above program so that only the lowercase alphabets (’a’ to ’z’) are printed. Work on the program to make it happen. Hint: The following statement can test if a character variable ch is lowercase.

```c
ch >= ’a’ && ch <= ’z’
```
Achieve Target 3 Cont 2

The program below contains a while loop that reads at most 128 numbers, and it terminates when the entered number is −1.

```c
#include <stdio.h>

void main() {
    float data = 0;
    int count = 0;

    printf("Enter at most 128 numbers below (-1 to finish)\n");
    while (count < 128) {
        scanf("%f", &data);
        if (data == -1)
            break;
        count++;
    }
}
```

Modify the above program so that the entered numbers are stored in an array of float. Remember to declare an array with an appropriate size. **Work** on the program to make it happen.

After you have finished the above, write another loop that prints the positive numbers stored in the array. Remember to declare loop counter variable for the second loop. **Work** on the program to make it happen.

Write another loop that detects and prints the largest number stored in the array. Remember to declare and initialise a temporary variable for detecting the largest number. **Work** on the program to make it happen.

The size of the array must be declared large enough for use. Trying to refer to array elements beyond the declared size will cause an error when running the program. In the programs, a guard should be used to prevent accessing an array beyond the specified size. In the solutions to the above exercises, the while loop serves as the guard; and in the program in the previous page, the for loop serves as a guard. **Learn** it.
Achieve Target 4

Let us look more closely at storing data into array. Study the following program.

```c
#include <stdio.h>

void main() {
    char charArray[256];
    int length;
    int i;
    
    printf("Enter a line of text (press Enter at the end):
    ");
    for (i=0; i<256; ) {
        scanf("%c",&charArray[i]);
        if (charArray[i] >= 'a' && charArray[i] <= 'z') {
            i++;
        } else if (charArray[i] == '\n') {
            length = i;
            break;
        }
    }
    /* printing the array elements */
    for (i=0; i<length; i++)
        printf("%c", charArray[i]);
}
```

The above program checks and stores lowercase alphabets only into the array. This selection of lowercase alphabets is carried out at the input stage, unlike previously at the printing stage. The key method used is the manipulation of the variable \(i\). The purpose of variable \(i\) is to index the array element for storing the coming character. Normally after each character is stored, the variable \(i\) should be increased by one so that it indexes the next element for storage. In the above program, an added condition is that the character is a lowercase alphabet (the statement with a red circle).

Rewrite the above program so that an entered character is stored when,

1. It is a uppercase alphabet with the exception of ‘C’, or;
2. It is a space.

**Work** on the program to make it happen.
Achieve Target 4 Cont

The following program is a solution to the previous Target exercise.

```c
#include <stdio.h>

void main() { 
    float anArray[128];
    int count = 0;
    int k;

    printf("Enter at most 128 numbers below (-1 to finish)\n");
    while (count < 128) {
        scanf("%f", &anArray[count]);
        if (anArray[count] == -1)
            break;
        count++;
    }

    for (k=0; k<count; k++) {
        if (anArray[k] > 0)
            printf("%f ", anArray[k]);
    }
    printf("\n");
}
```

This program stores all entered numbers into the array, but prints out only the positive array elements.

Modify the program so that only the positive numbers are stored in the array, instead of storing all numbers. Note that the variable `count` serves the same purpose as the variable `i` previously. **Work** on the program to make it happen.
Achieve Target 5

Arrays can be initialised in a similar way as initialising other variables.

```c
#include <stdio.h>

void main() {
    int intArray[] = {1, 3, 5, 7};
    float floatArray[] = {1, 3.2, 5.5};
    char charArray[] = {'a', 'e', 'i', 'o', 'u'};
    int i = 0;
    for (i=0; i<4; i++)
        printf("%d ", intArray[i]);
    printf("n");
    for (i=0; i<3; i++)
        printf("%f ", floatArray[i]);
    printf("n");
    for (i=0; i<5; i++)
        printf("%c ", charArray[i]);
    printf("n");
}
```

Output

```
1 3 5 7
1.000000 3.200000 5.500000
a e i o u
```

Note that that array size is not given – the square brackets have nothing inside. The array size can however be told from the initialising values. The array `intArray` is initialised with 4 integers, so the array size is 4. Similarly the array `floatArray` is initialised with three values, so the array size is 3. The hard-coded array sizes are used in the for loops for printing the array elements. Learn it.

Rewrite the above program so that the arrays are initialised with proper values so that the following is printed. Take care of the execution conditions of the loops.

Output

```
10 -12 3 4 5
+ - * /
```

Work on the program to make it happen.
Achieve Target 5 Cont

A pre-initialised array is useful for defining a data set. The following program uses a pre-initialised array to define a vowel letter set.

```
#include <stdio.h>

void main() {
    char charArray[] = {'a', 'e', 'i', 'o', 'u'};
    int numVowels = 0;
    int numAlpha = 0;
    char data;
    int i;

    printf("Enter a line of text (Enter to terminate):
");

    while (1) {
        scanf("%c", &data);
        if (data == '
')
            break;
        else if ((data >= 'a' && data <= 'z') || (data >= 'A' && data <= 'Z'))
            numAlpha++;
        for (i=0; i<5; i++) {
            if (data == charArray[i]) {
                numVowels++;
                break;
            }
        }
    }
    printf("Number of alphabets %d and vowels %d
", numAlpha, numVowels);
}
```

The main skeleton of the program uses a loop to detect how many alphabets there are in the input. This is rather standard application of loops. The new thing is the set of statements in blue colour. First, the program defines the set of vowels in the array `charArray`. Then for each character read in, the program compares the letter with each element in `charArray`. When a match occurs, then the variable `numVowels` (keeping the number of vowels) is increased by one. Learn it.

Note that the break statement within the `for` loop is not necessary but it can speed up the checking process. Once a match with an array element is found, there is no further need to check against the remaining array elements. So the `break` statement allows the checking process to finish for the current character.

The statements in the red box are equivalent to the following familiar `if` statement.

```
    if (charArray[i] == 'a' || charArray[i] == 'e' || charArray[i] == 'i' ||
        charArray[i] == 'o' || charArray[i] == 'u')
        numVowels++;
```

The program only detects vowels in lowercase. Modify the program so that it detects the uppercase vowels (‘A’, ‘E’, ‘I’, ‘O’, ‘U’) as well. Work on the program to make it happen.

If a defined data set is needed in your program, the use of array initialisation makes it easier to change than using the `if` statement alternative. Remember it.
Achieve Target 5 Cont 2

Let’s finish this Perform Exercise with an exercise that uses an array to define a data set.

```c
#include <stdio.h>

void main() {
    int number;
    int numPrimes = 0;

    printf("Enter integers from 1 to 10 (-1 to terminate):
"); 
    while (1) {
        scanf("%d", &number);
        if (number == -1)
            break;
        else if (number == 2 || number == 3 || number == 5 || number == 7)
            numPrimes++;
    }
    printf(\"Number of primes = %d\n\", numPrimes);
}
```

The data set required in this program is the set of prime numbers. The above program uses an `if` statement to detect the number of prime numbers entered.

Modify the above program so that,

1. The program uses an integer array to define a prime number data set from 1 to 20 (which are 2, 3, 5, 7, 11, 13, 17, and 19).
2. The `if` statement is replaced by a for/while loop that detects for a prime number.

Work on the program to make it happen.
Achieve Target 6

In the last 5 targets, you have worked hard on the following ideas. Ponder upon these ideas and remember what you have learned.

1. An array is a set of variables/elements of the same variable type. Each array has a size that indicates the number of array elements. The array elements are arranged in a linear sequence, each has an index or numbering. The first array element is always indexed or numbered zero (0). The following are examples of array declaration.

   ```
   int intArray[10]; /* an array of 10 int variables/elements */
   float floatArray[20]; /* an array of 20 float variables/elements */
   char charArray[5]; /* an array of 5 char variables/elements */
   ```

2. Each array element is no different from an ordinary typed variable. So each `int` array element is the same as an `int` variable. An array element is referred by the array name and its index or numbering. For example, the first element of `intArray` is referred by `intArray[0]`, and the third element of `charArray` is referred by `charArray[2]`.

3. Arrays are usually used with loops, whereby each array element is visited by each execution of loops. This is usually called array traversal – travel through the array. In array traversal loops, the loop counter is often used as the index variable of the array. As the loop counter changes, the index changes as well. Each array element is visited one after the other.

4. Arrays must be declared with a sufficiently large size. Accessing array elements beyond the declared size will cause error. Use array guard to prevent this from happening. An array guard is a condition that monitors the index variable of array and makes sure that the index cannot go beyond the declared size. Usually the loop execution condition is used as the array guard, but additional if statements could be used as well.

5. Array elements could be initialised with values. In this initialisation form, the values are arranged within a pair of curly brackets. The array size needs no explicitly declared in this form of array declaration.

   ```
   int intArray[] = {1, 2, 5, 8};
   ```

6. An explicit array size could also be specified in the initialisation form. The size, however, must be equal to or more than the number of initialising values. For an array size more than the number of initialising values, some array elements at the end will not be initialised. For example, in the last array declaration in the following, only the first four elements are initialised. The values of the last two elements are undefined.

   ```
   char charArray[3] = {'a', 'b', 'c'};
   int intArray[6] = {1, 2, 5, 8};
   ```

7. A common application of arrays is to store a pre-defined data set for comparison or matching. For example, we could define the set of vowels or prime numbers for detection purpose. The advantage is that we could make modification of the data set easily.