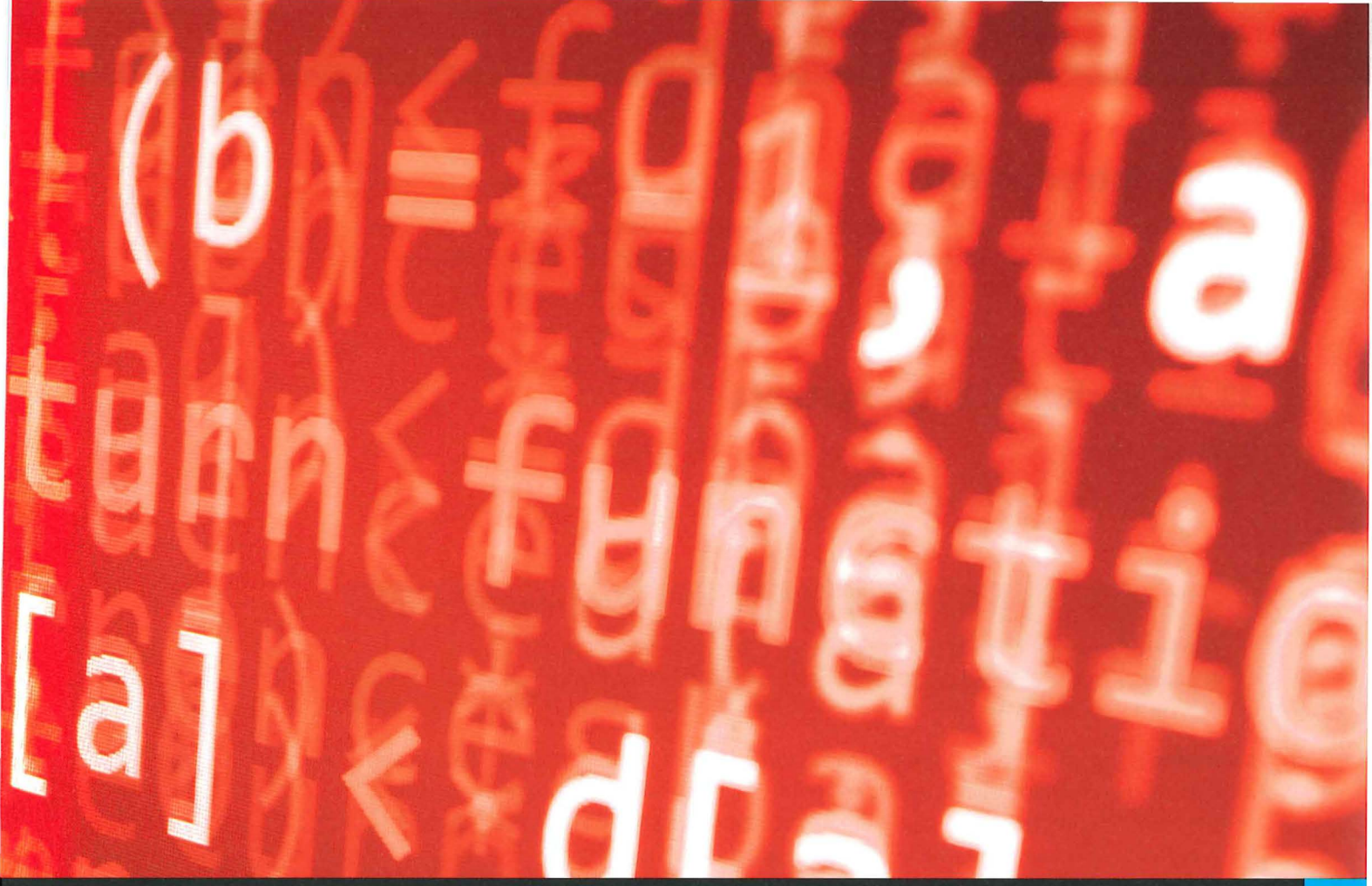

Chapter 11:
Programming Concepts

Learning objectives

By the end of this chapter you will:

- understand what variables and constants are and how to use them
- understand what the basic data types are and how to use them
- understand the basic principles of programming, such as sequence, selection, iteration, totalling and counting
- understand what predefined procedures and functions are
- be able to declare and use a one-dimensional array
- understand how to use a variable as an index in an array
- be able to read and write values into an array.

11.01 Introduction

In order to develop your skills as a programmer you need to understand the basic principles and structures involved in programming. The firmer an understanding of these basic principles you have, the better a programmer you are likely to be. So let us develop your understanding!

11.02 Variables and constants

SYLLABUS CHECK

Declare and use variables and constants.

KEY TERM

Variable – a named data location in a program for a value that can be changed.

Constant – a named data location in a program for a value that does not change.

A **variable** is a storage location. It is a named value that contains data that can be changed throughout the execution of a program. A variable will be created and assigned a value at a certain point in a program. This means the variable can then be used repeatedly throughout the program, even changing its value if it is reassigned a new one.

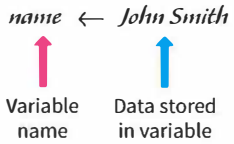

Figure 11.01 Variable information

Variables can be given almost any name, but sensible names should be given that are connected to the data that will be stored in them. It is also good practice to avoid putting spaces in variable names as it can cause some issues.

At the start of a program we could assign the data 'John Smith' to our variable 'name'. However during our program we may assign various other values to the variable, such as 'Alice Jones' or 'Aditya Bhambra'. We keep the name of the variable as 'name': it is the data that is stored in the variable that changes.

A **constant** is also a storage location. It is a named location that contains a value that we don't want to change during the running of the program, e.g. we want the value stored there to remain constant. Constants can be very useful in programming: you only need to change the value where the constant is defined and it will change in every other place it is used within the program. We could have a constant called 'name' that stores the data 'John Smith' and it will store the same data throughout the program.

The difference between a variable and a constant is that the value stored in a variable can be changed throughout a program, but a value stored within a constant will remain the same throughout the program.

11.03 Data types

SYLLABUS CHECK

Understand and use basic data types: Integer, Real, Char, String and Boolean.

A variety of data types can be stored in a variable or a constant including strings, integers, real numbers, characters and Boolean data. Normally a variable or a constant will be assigned a suitable data type for the data that it will store.

String

A string is the name given to a series of characters or symbols. We may more commonly call a string 'text'. In some programming languages a variable can be stored as a string by putting quotation marks around the data:

```
name ← "John Smith"
```

Sometimes we may need to convert another variable into a string. Programming languages will normally have a built-in method to convert variables between data types.

Numbers

An integer is a whole number that has no decimal points, for example 100 would be an integer but 100.1 would not. Integers are used in programming to store data that would never need to be a decimal or a fraction. For example, a program that stored the year that a person was born would use an integer for this. A year, such as 2001, will never be a decimal number.

A real number does have decimal points. An example of a decimal number would be 3.142. Real numbers can also be referred to as 'float'. They are used to store data that could be a decimal or a fraction. For example, a program that stored the test scores of a class of students could have decimal numbers, as a student may gain half a mark for a question. A mark such as 9.5 out of 10 would need to be stored as a real number.

Numbers stored in a variable can be defined as an integer or real. A variable can also be converted to an integer or real if it is set as another data type initially. How this is done will depend on the programming language used.

Character

Character is a data type that holds just one character. Even though a string can hold just one character, it is more efficient to store a single character in a character data type. This is because a string may have a minimum number of bytes allocated for that data type, for example 4 bytes, so a single character would still be stored in 4 bytes of data. In contrast, a character data type will store the single character in 1 byte of data. An example of using a character could be storing M or F to represent if a person is male or female.

Boolean

A Boolean data type is one that will hold two values only, such as true/false or yes/no. When we know that an instance only has two values, Boolean is the best data type to use.

An example of Boolean data would be if a program stored data about whether a student has passed an exam or not. A variable named 'pass' could store the value yes or no.

11.04 Basic principles of programming

SYLLABUS CHECK

Understand and use the concepts of sequence, selection, repetition, totalling and counting.

KEY TERM

Algorithm – a set of instructions to solve a problem.
Sequence – instructions that occur one after the other in a particular order.
Executed – a program that is run.
Selection – making a choice or a decision in a program.
Iteration – repeating instructions in a program.
Condition – a state in a program that will be met or not met.

Sequence

In an **algorithm** there are a number of basic programming principles that you need to be aware of. Having a good understanding of these principles will improve your programming skills.

In an algorithm, each instruction is written one after the other. This is a called a **sequence**. Most computers handle instructions one at a time, so most programs are written as a sequence. A sequence can contain any number of steps but they have to be **executed** in the correct order:

Input first number
Store the first number
Input the second number
Add first number to the second number
Output the result
Stop the program

If any of these steps were written out of order then the result of the algorithm may not be the same. Therefore, sequence is a vital principle of programming.

Selection

When creating a program it may be necessary to create a way to follow different paths through the program. Creating the ability to choose the path to follow in a program is called

selection. Selection is the process of making a decision in a program. In a selection a question is asked and then the correct path is taken depending on the answer.

```
age ← USERINPUT
IF age >18 THEN
    PRINT "You are an adult"
ELSE
    PRINT "You are not an adult"
```

IF...THEN...ELSE is a common structure used to create selection in a program.

Iteration

Sometimes in a program there will be some instructions that need to be repeated. To repeat instructions we put them in a loop and this is referred to as **iteration**. There are two main types of loop that can be created, a counting loop and a condition loop. A counting loop repeats a set of instructions a set number of time. This loop will print 'Happy' five times:

```
FOR 5 loops:
    PRINT "Happy"
ENDFOR
```

A condition loop repeats a set of instructions until a condition is met:

```
Me ← "Happy"
REPEAT:
    PRINT Me
UNTIL Me = "Sad"
```

This will keep printing 'Happy' until the variable Me becomes 'Sad'.

A programmer has to make the decision which kind of iteration loop to use to repeat the instructions. If they want to repeat a set of instruction a defined number of times then they would use a counting loop. If they want to repeat a set of instructions until a **condition** is met or a condition stops being met then they would use a condition loop.

Totalling

It may be necessary to keep a running total in a program. For example, if we had a program that was a quiz, each time the user gets a question correct the score is updated. A variable to store the score would be created. This variable may originally be set to 0. Each time a user gets a question correct 2 is added to the score. This is done by taking the current value that is in the variable and adding 2 to it.

We would define the variable:

```
score ← 0
```

Then we would add a value of 2 to the score each time a question is correct:

```
score ← score + 2
```

By referring to the name of the variable in the calculation, it will take the value currently stored in 'score' and add 2 to it.

Counting

It may be necessary to keep a count of the amount of times something occurs in a program. For example, if we had a program that allowed a teacher to enter marks for a class, we may want to count how many marks they enter. Counting works in a very similar way to totalling. A variable to store the count would be created. This variable may be originally set to 0. Each time a process occurs 1 is added to the count. This is done by taking the current value that is in the variable and adding 1 to it. For example:

We would define the variable:

```
count ← 0
```

We then add a value of 1 to the count each time a process (such as entering a mark) occurs:

```
count ← count + 1
```


11.05 Using predefined routines

SYLLABUS CHECK

Use predefined procedures/functions.

KEY TERM

Predefined function – a pre-programmed set of instructions that return a value.

Library – a store of pre-programmed instructions that can be imported into a program.

Predefined procedure – a pre-programmed set of instructions that do not return a value.

Most programming languages have **predefined functions** that you can use as a programmer. These are functions that cover tasks that programmers commonly use such as converting integers to strings and file handling. They can be built into the language or imported from a **library**. These functions are great because we don't have to code them ourselves and they are pre-tested so we know they work.

Predefined procedures are the same except that they do not return a value like a function does. An example of a predefined procedure could be a 'clear the screen' command that would clear whatever is currently displayed when running the program.

11.06 Using an array

KEY TERM

Array – a store of data values that are all related and of the same data type.

Element – an individual data location in an array.

What is an array?

SYLLABUS CHECK

Declare the size of one-dimensional arrays, for example: A[1:n].

Show understanding of the use of a variable as an index in an array.

An **array** is a way of storing data that is all related and of the same type e.g. a list of the names of the students in your class could be stored in an array. Each item that is stored in an array can be accessed by referring to its location within the array.

If we wanted to store the highest scores that a player achieves in a game, we could do this in an array. An example of an array storing high scores would be:

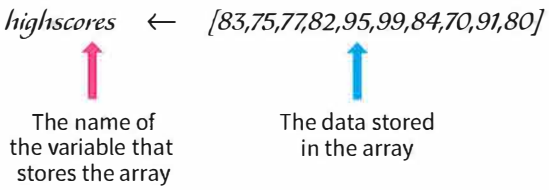

Figure 11.02 Example of an array storing high scores

Each individual item in the array has a location that can be pointed to. One important thing to note is that in most programing languages the locations start at 0 and not 1. For example the data in location 3 of the array in Figure 11.02 would be 82. Each of the individual items in an array is called an **element**.

Creating an array

In order to be able to store the high scores in an array we would need to carry out the following steps:

1. Give the variable that will store the array a name, for example highscores.
2. Set the number of elements the array will hold. This is called specifying the dimension of the array.
3. Define the type of data the array will hold. This is called specifying the data type. This step is not needed in some programming languages.

One important thing to note is that once we have defined the size (dimension) of an array, it normally cannot be changed.

Once we have created an array we can then write data to the array and read data from it.

Reading to and writing from arrays

We can ask a program to **read** certain elements in an array. To do this we would need to ask the program to output the data in location in an array. The program will read the data in this location in the array and output it for us to see. The following pseudocode outputs data elements from the array in Figure 11.02:

```
PRINT highscores(3)
PRINT highscores(8)
PRINT highscores(0)
```

KEY TERM

Read – outputting data from an array so we can see it.

Write – inputting data into an array to store it.

The output would be the fourth, ninth and first elements: 82, 91 and 83.We can ask a program to **write** elements to an array. To do this we state the data location in the array that we want to write data to, followed by the data. The following pseudocode writes the high score of 90 to the second element location in the array:

```
highscores(1) ← 90
```

If the array already held a value in this element that value would be overwritten with the new value. This would mean that the current value in location 1 of the array in Figure 11.02 (75) would be overwritten with 90.

To learn more about pseudocode, see Chapter 12.

TEST YOURSELF

If we want to overwrite the first element in the high scores array with the data '99', what would the pseudocode be for this?

We can also use a variable to read data from or write data to an array. We can do this by replacing the element number with the variable name. An example of reading data from our high scores array, using a variable, would be:

```
PRINT highscores(count)
```

If the variable 'count' held the value 3, this would output the data from the fourth element in the array (82).

An example of writing data to our high scores array, using a variable, would be:

```
highscores(count) ← 75
```

If the variable 'count' still held the value 3, this would overwrite the data in the fourth element in the array (82) with the value 75.

Reading to and writing from arrays using loops

To read or write multiple values with an array we can use a FOR ... TO ... NEXT loop.

SYLLABUS CHECK

Read or write values in an array using a FOR ... TO ... NEXT loop.

A FOR ... TO ... NEXT loop will read values from or write values to an array for a set number of times:

```
FOR count ← 0 TO 3:
    PRINT highscores(count)
NEXT count
```


The 'count' variable used refers to the element number that is currently being read. It begins at 0 and when it gets to NEXT count it will increase by 1, till it gets to 3. Therefore this loop would read and output elements 0, 1, 2 and 3. With our high scores array, this loop would output the values: 83, 75, 77 and 82.

An example of using a FOR ... TO ... NEXT loop to write values to an array would be:

```
FOR count ← 0 TO 3:
    highscores(count) ← USERINPUT
NEXT count
```

With our high scores array, this FOR... TO... NEXT loop would write the value input to the relevant element. It would start with writing the first input to element 0 and keep repeating till it had written to element three. If we input the numbers 75, 77, 90 and 99 our high scores array would now look like this:

```
highscores ← [75,77,90,99,95,99,84,70,91,80]
```

The program does not necessarily start at element 0 each time when reading and writing data values. If the loop started with:

```
FOR count ← 2 TO 7:
```

The program would read or write element 2 through to element 7.

To learn more about FOR ... TO ... NEXT loops, see Chapter 12.

Why do we use arrays?

Arrays are very useful and have several advantages for programmers:

- They store many data values under one name. This means we do not have to use or remember many variable names.
- We can easily find an item of data by specifying its element number.
- We can easily read or write many values by using a FOR ... TO ... NEXT loop.

Arrays also have some disadvantages:

- Once defined, the size of the array cannot normally be changed. This can be wasteful of memory if we don't use all the array's elements to hold data.
- Arrays can only hold data of one data type (the type specified when the array is defined).

Multi-dimensional arrays

The arrays we have been using are called one-dimensional arrays. This is because they only contain one set of data. Arrays can be of more than one dimension. A two-dimensional array is an arrays that holds two sets of data. An example of a two-dimensional array would be:

```
highscores2 ← [75,77,90,99,95][74,98,90,88,78]
```

Multi-dimensional arrays are not part of the syllabus that you need to know. This information has been included to help you to understand the concept of an array and that they can be expanded to hold more than one set of data.

Summary

- A variable is a storage location that contains data that can be changed throughout the execution of a program.
- A constant is a storage location that contains a value that will never change.
- A string is a series of characters or symbols.
- An integer is a whole number with no decimal point.
- A real number is a number that does have a decimal point.
- Character is a data type that holds just one character.
- A Boolean data type is one that will hold two values only.
- A sequence is a set of instructions that occur one after the other in a specific order.
- Selection is the process of making a decision in a program.
- Iteration means repeating instructions in a loop. A counting loop repeats a set number of times and a condition loop repeats till a condition is met.
- Totalling is the process of keeping a running total of values in a program.
- Counting is the process of keeping a running count of how many times something happens in a program.
- A predefined routine is a pre-programmed set of instructions that can be used in a program. They are often stored in a library or built into the programming language:
  - A predefined function will return a value.
  - A predefined procedure will not return a value.
- An array is a way of storing data that is all related and of the same type. Each item that is stored in an array can be accessed by referring to its location within the array.
- Each of the individual items in an array is called an element.
- In order to create an array we need to name the variable that will store it, set the number of elements we want to hold and define the data type.
- An advantage of arrays is that they can store many data values under one name.
- A disadvantage of arrays is that once the size of the array is defined it cannot be changed. This can be wasteful of memory if we don't use all the array's elements to hold data.

Exam-style questions

1 Explain the difference between a variable and a constant. Give an example of when you would use each. (2 marks)

2 A programmer would like to store the following data:

- **a** the user's name (1 mark)
- **b** the user's age (1 mark
- **c** the user's gender (1 mark
- **d** the user's quiz score. (1 mark

Give the most suitable data type for each item of data they want to store.

3 A teacher wants to work out the average test score for a class. They want to count and total the number of marks they enter to calculate the average.

- **a** Explain how they would do this. (5 marks)
- **b** The teacher wants to enter marks till they have no more marks to enter. They do not currently know how many marks they will need to enter. Which loop structure would they use to repeat the process of entering marks and why? (2 marks)

4 Explain why sequence is important when creating a program. (1 mark)

5 State one reason why a programmer would use an array. (2 marks)

6 A teacher wants to create a program that will allow them to write exam scores for a class to an array. There are 15 students in the class. Write a suitable pseudocode solution that will allow the teacher to do this. (4 marks)

7 When creating an array, what are the two main things that we need to do? (2 marks)