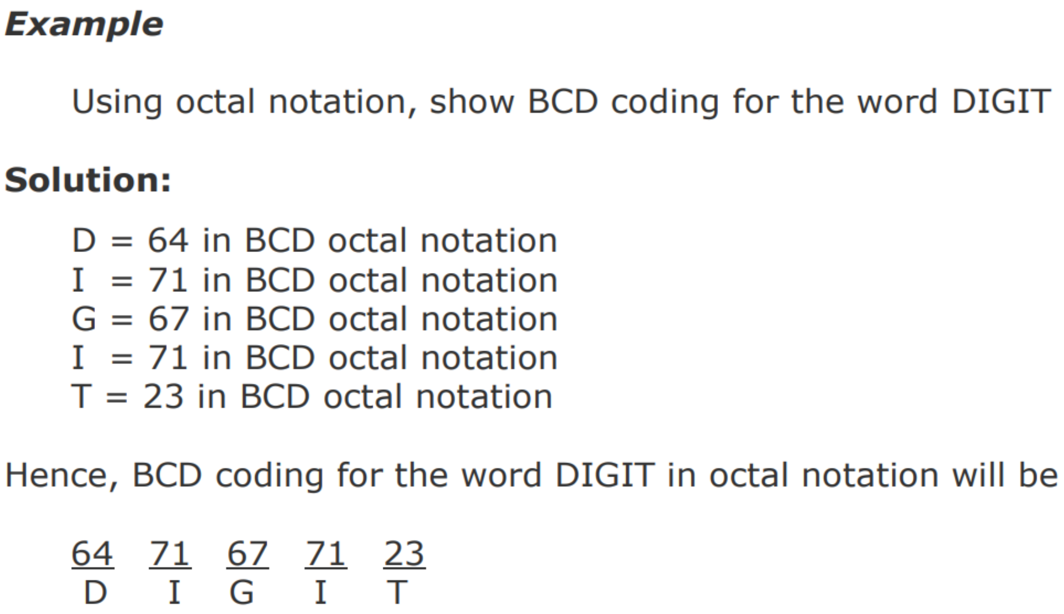
Computer Codes and Data Structure

Computer Codes

- Computer codes are used for internal representation of data in computers
- As computers use binary numbers for internal data representation, computer codes use binary coding schemes
- In binary coding, every symbol that appears in the data is represented by a group of bits
- The group of bits used to represent a symbol is called a byte
- As most modern coding schemes use 8 bits to represent a symbol, the term byte is often used to mean a group of 8 bits
- Commonly used computer codes are BCD, EBCDIC, and ASCII

BCD

- BCD stands for Binary Coded Decimal
- It is one of the early computer codes
- It uses 6 bits to represent a symbol
- It can represent 64 (2^6) different characters



Example

Using octal notation, show BCD coding for the word DIGIT

Solution:

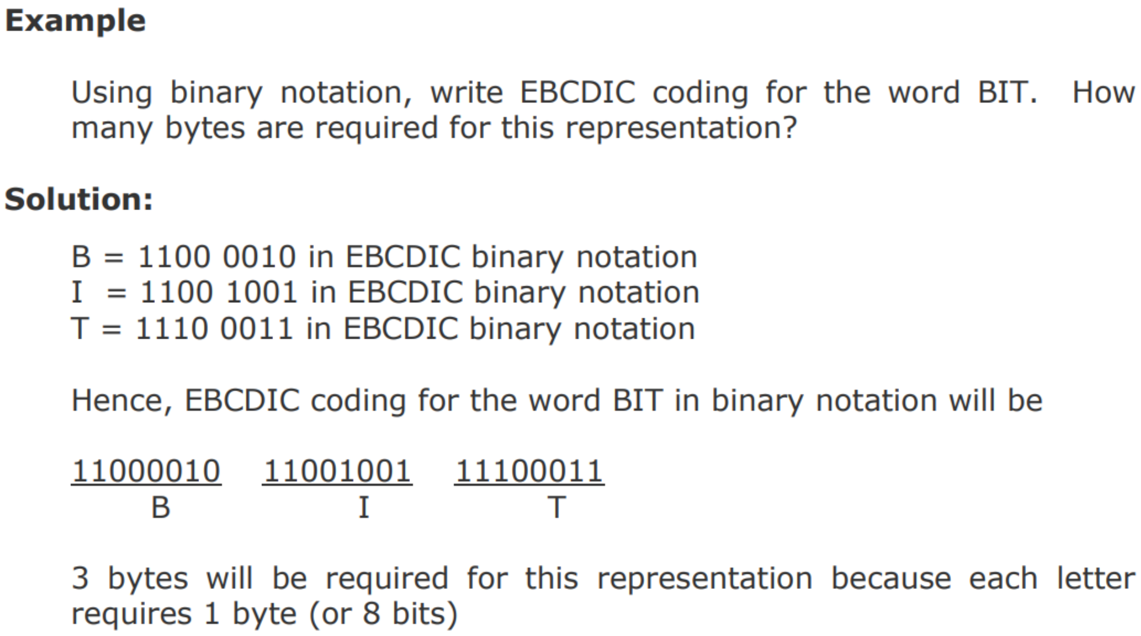
D = 64 in BCD octal notation
I = 71 in BCD octal notation
G = 67 in BCD octal notation
I = 71 in BCD octal notation
T = 23 in BCD octal notation

Hence, BCD coding for the word DIGIT in octal notation will be

$\underset{D}{\underline{64}}\quad \underset{I}{\underline{71}}\quad \underset{G}{\underline{67}}\quad \underset{I}{\underline{71}}\quad \underset{T}{\underline{23}}$

EBCDIC

- EBCDIC stands for Extended Binary Coded Decimal Interchange Code
- It uses 8 bits to represent a symbol
- It can represent 256 (2^8) different characters



Example

Using binary notation, write EBCDIC coding for the word BIT. How many bytes are required for this representation?

Solution:

B = 1100 0010 in EBCDIC binary notation
I = 1100 1001 in EBCDIC binary notation
T = 1110 0011 in EBCDIC binary notation

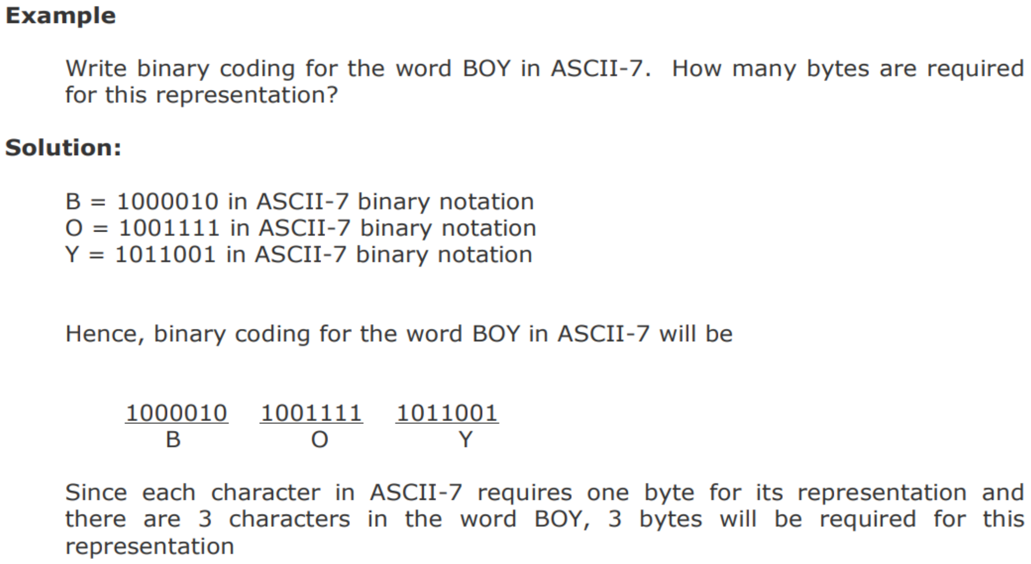
Hence, EBCDIC coding for the word BIT in binary notation will be

$\underset{B}{\underline{11000010}} \quad \underset{I}{\underline{11001001}} \quad \underset{T}{\underline{11100011}}$

3 bytes will be required for this representation because each letter requires 1 byte (or 8 bits)

ASCII

- ASCII stands for American Standard Code for Information Interchange.
- ASCII is of two types – ASCII-7 and ASCII-8
- ASCII-7 uses 7 bits to represent a symbol and can represent 128 (2^7) different characters
- ASCII-8 uses 8 bits to represent a symbol and can represent 256 (2^8) different characters
- First 128 characters in ASCII-7 and ASCII-8 are same



Example

Write binary coding for the word BOY in ASCII-7. How many bytes are required for this representation?

Solution:

B = 1000010 in ASCII-7 binary notation
O = 1001111 in ASCII-7 binary notation
Y = 1011001 in ASCII-7 binary notation

Hence, binary coding for the word BOY in ASCII-7 will be

$\underset{B}{\underline{1000010}} \quad \underset{O}{\underline{1001111}} \quad \underset{Y}{\underline{1011001}}$

Since each character in ASCII-7 requires one byte for its representation and there are 3 characters in the word BOY, 3 bytes will be required for this representation

Data Structure

A data structure uses a collection of related variables that can be accessed individually or as a whole. In other words, a data structure represents a set of data items that share a specific relationship. We discuss three data structures in this chapter: arrays, records, and linked lists. Most programming languages have an implicit implementation of the first two. The third, however, is simulated using pointers and records.

1- An array

An array is a sequenced collection of elements, of the same data type. We can refer to the elements in the array as the first element, the second element, and so forth until we get to the last element. If we were to put our 100 scores into an array, we could designate the elements as scores [1], scores [2], and so on. The index indicates the ordinal number of the element, counting from the beginning of the array. The elements of the array are individually addressed through their subscripts (Figure 9.1). The array as a whole has a name, scores, but each score can be accessed individually using its subscript.

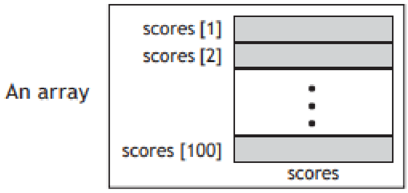


Figure (9.1) An array with index

We can use loops to read and write the elements in an array. We can also use loops to process elements. Now it does not matter if there are 100, 1000, or 10000 elements to be processed—loops make it easy to handle them all. We can use an integer variable to control the loop, and remain in the loop as long as the value of this variable is less than the total number of elements in the array (Figure 9.2).

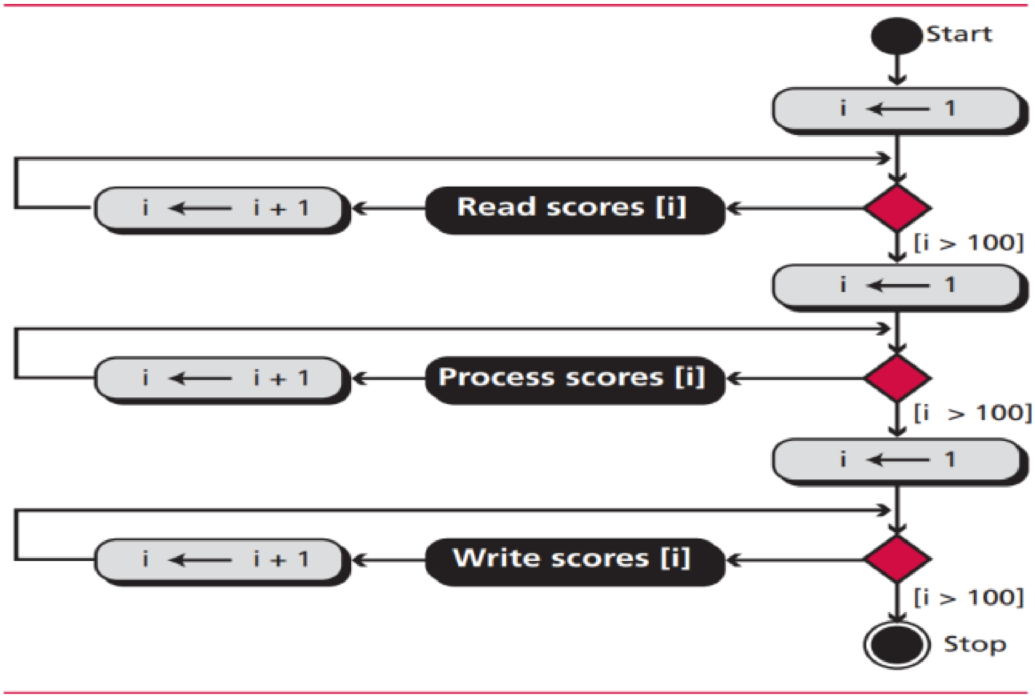


Figure 9.2Processing an array

Example 1

Compare the number of instructions needed to handle 100 individual elements and the array with 100. Assume that processing each score needs only one instruction.

Solution

❑ In the first case, we need 100 instructions to read, 100 instructions to write, and 100 instructions to process. The total is 300 instructions.

❑ In the second case, we have three loops. In each loop we have two instructions, giving a total of six instructions. However, we also need three instructions for initializing the index and three instructions to check the value of the index. In total, we have 12 instructions.

2- RECORDS

A record is a collection of related elements, possibly of different types, having a single name. Each element in a record is called a field. A field is the smallest element of named data that has meaning. A field has a type, and exists in memory. Fields can be assigned values, which in turn can be accessed for selection or manipulation. A field differs from a variable primarily in that it is part of a record.

Figure 9.3 contains two examples of records. The first example, fraction, has two fields, both of which are integers. The second example, student, has three fields made up of three different types.

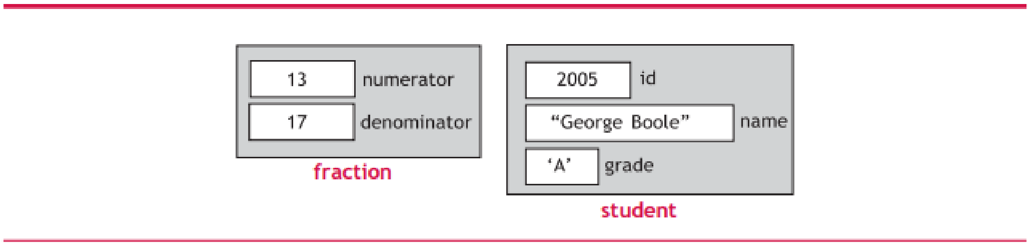


Figure (9.3) A record

Example 2

The following shows how the value of fields in Figure 9.3 are stored:

student.id ← 2005 student.name ← "G. Boole" student. grade ← 'A'

3- LINKED LISTS

A linked list is a collection of data in which each element contains the location of the next element—that is, each element contains two parts: data and link. The data part holds the value information: the data to be processed. The link is used to chain the data together, and contains a pointer (an address) that identifies the next element in the list. In addition, a pointer variable identifies the first element in the list. The name of the list is the same as the name of this pointer variable. The link in each element, except the last, points to its successor. The link in the last element contains a null pointer, indicating the end of the list. We define an empty linked list to be only a null pointer. Figure (9.4) also shows an example of an empty linked list.

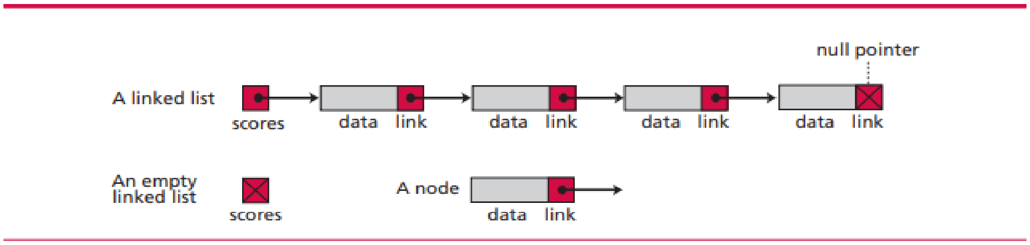


Figure (9.4) empty linked list.