Chapter 5

Data structures

Arrays, strings, and records

1. Introduction

In the previous chapters, we have seen that all data manipulated in algorithms are of simple types.

A simple type variable corresponds to a single piece of information, such as an integer, real number, boolean, or character.

In contrast to simple types, structured types are made up of basic types or other declared types. Also known as complex types, they represent a collection of multiple values.

Another term often used in algorithmics is the notion of data structure. A data structure is a particular way to store and organize data for more efficient and easier processing.

Data structures come in different types, each with their own strengths and weaknesses.

In this chapter, we will focus on three of the most common data structures:

- Arrays are a simple data structure that stores data in a linear order.
- Multidimensional arrays are a generalization of arrays that can store data in a multidimensional order.
- Character strings are a data structure that stores a sequence of characters.

We will also briefly discuss other data structures, such as records, files, and lists. These data structures will be covered in more detail in later chapters.

2. The array type

1. Definitions

An array is a finite set of objects of the same type. It can also be defined as a collection of homogeneous data, accessible by an index. Or, as a data structure that allows for the same processing on data of the same nature.

The number of elements in an array defines its cardinality, which is ***constant*** during the execution of the algorithm. The number that serves to identify each element of the array is called ***an index***. The type of the elements defines the type of the array. In other words, an array is a type itself based on another type.

The advantage of arrays is that they allow you to designate a set of values using a single identifier.

In fact, a single variable of array type will represent an entire collection of values, instead of one for simple types.

2. Declaration:

Declaring an array requires specifying its cardinality and the type of its elements:

TYPE ‹Array_Name› = array [‹size›] of ‹type_of_elements› ;

Where:

- **‹ Array_Name ›:** is the identifier designating the name of the array type.
- **‹size›:** enclosed in square brackets, indicates the number of elements the array can hold. This size is determined when the array is declared and cannot be modified afterward.
- **‹ type_of_elements ›:** Any simple or structured type is allowed, except files.

Examples

Example 1

```
Type tab = array [20] of integer;
Var T1: tab ;
    T2: array [5] of char;
```

Example 2

```
Type colour = (green, red, white, blue);
      stock = array [colour] of integer;
Var S: stock;
```

Example 3

```
Const card = 10 ;
Type table = array [card] of real ;
Var Table1 : table ;
```

3. Accessing array elements

To access an element in an array, the name of the array is followed by the index of the element, enclosed in square brackets. The index of an element is a number that indicates its position in the array. The first element has an index of 0, the second element has an index of 1, and so on. access can be described as follows:

‹Array_Name› [index]

Examples

Example 1 :

```
Var T : array [5] of integer ;
….
T[1] ← 5 ;
i ← 1 ;
read (T[i]);
write (T[i+3]) ;
….
```

Example 2

```
Type day = (SAT, SUN, MON, TUE, WED, THU, FRI) ;
      State = (free, occupied) ;
Var agenda : array [day] of state;
…
agenda [SAT] ← occupied;
...
```

agenda [SAT] is the first element of the array.

Remark

The index of an array with an enumerated cardinality can only take the values listed in the enumeration, otherwise the value of the index must be between 0 and size-1.

4. In C language:

‹type_of_elements› ‹Array_Name› [size];

Example : int T[20] ;
This declaration creates an array of 20 integers, called T.

So, declaring an array, in C language, consists of three parts:

- The type of the array elements. In this example, the type is int.
- The array identifier. In this example, the identifier is T.
- The array size, always specified between square brackets. In this example, the size is 20.

5. Memory representation

In memory, an array is stored in a contiguous block, with each element following the previous one. In other words, the first element is followed by the second, which is directly followed by the third, and so on. As a result, the array address is the address of its first element.

6. Application Exercises

Exercise 1

Write an algorithm to initialize a 20-element integer array and display its contents.

Exercice 2

Write an algorithm that fills an integer array as follows:

- The first two elements are given by the user.
- Each subsequent element is the sum of the two previous elements.

Exercise 3

Write an algorithm to enter a 20-element integer array, calculate their sum and average, and display the results.

Exercise 4

Write an algorithm to enter an array of N real numbers, calculate the number of positive and negative values, and display the results.

Exercise 5

Write an algorithm that finds the first occurrence of a value ***V*** in a 10-integer array ***T***. If the value ***V*** does not exist in the array, the algorithm should display ***-1***.

Exercise 6

Write an algorithm that inputs a 10-element integer array ***T***, then calculates and prints its maximum value.

3. Multidimensional Arrays

Consider the problem of modeling the grades of a class of 30 students in 10 subjects.

One solution is to use an array for each student where each element represents the grades of a student, that is, each element of the array corresponds to an array of grades. We can describe it algorithmically as follows:

```
Const nb_stud=30; nb_grade=10;
Type Grades = array [nb_grade] of real ;
     Students = array [nb_stud] de Grades ;
Var S: Students;
```

To access an element of this table, for example, to enter the jth grade of the ith student, we use: Read (S[i] [j])

Another solution is to use an array where each element represents the grades of a subject, that is, each element of the array corresponds to an array of grades of all the students in a subject. It can be described algorithmically as follows:

```
Const nb_stud=30; nb_grade=10;
Type Subject = array [nb_stud] of real;
     Grades = array [nb_grade] of Subject;
Var G: Grades;
```

Assume that each student has 4 grades (exam, TD, TP, rattrapage) per subject. To represent all grades in this case, each element of the array S (or G) must be an array of 4 elements, and to enter the kth mark for the ith student's jth subject, we must use: read(S[i] [j] [k]);

And so on ...

1. Definitions

One-dimensional arrays, also known as vectors or single-dimensional array, are the simplest type of array. However, they are not always sufficient for solving complex problems. In algorithms, two-dimensional arrays, also known as matrices, can be used to represent data that has two dimensions. Three-dimensional arrays, also known as hypermatrices, can be used to represent data that has three dimensions. And so on.

2. Declaration:

Arrays of higher dimensions can be declared in a similar way to one-dimensional arrays. However, instead of a single index, several indices must be specified, one for each dimension.

For example, the following declaration defines a two-dimensional array of real numbers:

```
Const nb_stud=30 ; nb_grade=10 ;
Type Grades = array [nb_stud, nb_grade] of real ;
Var G : Grades ;
```

This array can be used to store the grades of all students in all subjects. The first index represents the student, and the second index represents the subject.

3. Accessing Multidimensional array elements

To access an element in a multi-dimensional array, the name of the array is used, followed by the indices. For example, to access the jth grade of the ith student in a two-dimensional array G, the following syntax is used: read(G[i, j]).

If we use an ***n***-dimensional array ***T***, each element is referenced by ***n*** indices.

T [ind1, ind2, ind3, …, indn]

4. In C language :

> *‹ type_of_elements › ‹ Array_Name› [size1] [size2] … [sizen];*

sizei is the size of the dimension ***i***. It must be a constant defined before or at the time of declaration.

An element of an n-dimensional array ***T*** is identified by its ***n*** indices.

T [ind1] [ind2] … [indn]

5. Application Exercises

Exercise 1

Write an algorithm to initialize an integer matrix, count the number of even and odd integers in the matrix, and display the results. Zero is considered as an even integer.

Exercise 2

write an algorithm to initialise an identity matrix of size $N * N$. The algorithm must then display the contents of the matrix.

Exercise 3

Given a matrix ***A*** of dimensions ***NxM*** (2x3), write an algorithm to input ***A***, calculate its transpose ***AT*** (of dimensions ***MxN***), and display it.

The transpose of a matrix is obtained by swapping its rows and columns.

Example: A =$\begin{bmatrix} 1 & 2 & 3 \\ 11 & 22 & 33 \end{bmatrix}$, *AT*=$\begin{bmatrix} 1 & 11 \\ 2 & 22 \\ 3 & 33 \end{bmatrix}$

Exercise 3

Let A be an n-by-m matrix and B be an m-by-p matrix. The product of A and B is a matrix C that is n-by-p in size. Each element of C is calculated as the sum of the products of the corresponding elements of A and B, as follow:

$$c_{i,j} = \sum_{k=0}^{m-1} a_{i,k} b_{k\,j}$$

Write an algorithm that allows to enter the two matrices A and B, calculate and display their product.

4. Strings

1. Definitions

Strings are used to store text data in a computer program, such as names, addresses, sentences, etc.

A string is an ordered set of characters, which can be represented ***by an array of characters***. Strings are composed of characters. Each character, which is a symbol, has a position in the string, called its ***rank***. The rank of a character in a string is equivalent to the index of an element in an array.

A character can be one of four types:

- Alphabetic : a letter, lowercase or uppercase.
- Numeric : a digit.
- Punctuation : a punctuation mark.
- Symbolic : any other symbol.

Examples

- The string "Baobab" consists of the **characters** "B", "a", "o", "b", "a" and "b".
- The string "123" consists only of **digits**, but it should never be confused with the numeric value 123.
- A string of "" represents an **empty string**.
- A string of " " is a string **of one character**, which is the space character. *It is important to not confuse an empty string with a string containing a single space character,*

2. Declaration

In algorithms, the string type declaration is as follows:

```
TYPE <String_Name> = string [length] ;
```

Where: ***length*** specifies the maximum size of a variable of this type.

The memory representation of a string adds one byte to store its length. This length is the number of characters that will be displayed by a write instruction. If the length is not specified, a default length of 255 characters is used.

Example

Type nom = string [20] ;

Var name: nom ;

3. Operation on Strings

Concatenation of Strings

Concatenation is an operation that combines two or more strings into a single string, by placing the characters of each string one after the other.

In algorithmics, the concatenation operator is the symbol +.

Comparing Two Channels

To compare two strings, we compare the string characters starting with the first character of each string:

- *the first character of the first string is compared with the first character of the second string,*
- *the second character of the first string is compared with the second character of the second string,*
- *And so on...*

Examples

- "**b**aobab" < "**s**apin" because the ASCII code of "b" is less than the ASCII code of "s".
- "ba**o**bab" > "ba**n**ania" because the ASCII code of "o" is greater than the ASCII code of "n". The comparison cannot be made on the first two characters because they are identical.
- "1999" > "1998" because the ASCII code of "9" is greater than the ASCII code of "8". The comparison cannot be made on the first three characters because they are identical.
    - Note that the comparison is not between numerical values, but between characters.
- "**3**33" > "**1**230" because the ASCII code of "3" is greater than the ASCII code of "1".

- "333" < "333**0**" because the second string is longer than the first string. The comparison cannot be made on the first three characters because they are identical.
- "**B**aobab" < "**b**aobab" because the ASCII code for "b" is greater than the ASCII code for "B".

4. In C language:

In the C programming language, there is no special data type for strings. A string is simply a one-dimensional array of characters. However, the language provides notations and special functions for manipulating arrays of characters.

The declaration of a string in C is done as follows:

> *Char ‹String_Name› [length] ;*

Notes

- The internal representation of a character string is terminated by the '\0' symbol. Therefore, to store a character string of n characters, n + 1 bytes must be allocated.
- Unfortunately, the C compiler does not check whether we have reserved a byte for the end-of-string symbol; the error will only be noticed when the program is run ... The advantage of the end-of-string character is that it allows variable-size strings to be stored in a fixed-size array. We have seen that the size of an array is constant and cannot be changed. But you can change its contents, in other words, you can change the characters it contains. In particular, you can shorten or lengthen the text. It is therefore necessary to use '\0'.
- A string can also be declared as a pointer to char.
- In addition to the traditional method of declaring a character array, there is a specific method for declaring a string.
  - This method uses quotation marks, as follows: char s[] = "chaine";
  - Note that with this method, you do not need to include a terminating null character (\0).

- This operation is performed automatically by the compiler, as the quotation marks unambiguously indicate that the variable is a string.
- The scanf and printf functions manipulate strings using the format code ***%s***.

Example :
```
char ch[20];
scanf("%s",ch);
printf("The string is %s.",ch);
```

5. Application Exercises

Exercise 1

Write a program to enter a string of characters and then calculate the length of the string, i.e. the number of characters typed by the user.

Exercise 2

Write a program that enters a character string chaine1[10], then copies this string to another string chaine2[10].

Exercise 3

Write a program to enter two strings stored in the arrays chaine1[10] and chaine2[10], concatenate these two strings to obtain a third string chaine3, and finally display chaine3.

5. Records

1. Introduction

A warehouse management system is to be developed. Each product is described by its name, code, unit price, and available quantity. The system must provide the following functionalities:

- Input product information.
- Display product information.
- Input a product array.
- Display a product array.
- Display a list of nonexistent products.

- Add a new attribute for the expiration date, and another for the product category (food, cleaning, or accessory). Display a list of expired products.
- Finally, separate the products into three arrays: tab_ food, tab_ cleaning, and tab_ accessory.

The following questions need to be answered:

1. How to represent a product in this program?
2. How to represent its category?
3. How to represent its expiration date?

Arrays and matrices are data structures that allow you to group elements of the same type. However, it is often necessary to group elements of different types.

For example, in a bank, it is necessary to store information about customers, such as their number, name, address, and zip code.

This information is of different types, so it is necessary to use a data structure that is adapted.

1. Definition

The record type allows to define a structure of variables that groups elements of different types.

The components of a record are called ***fields***.

2. Declaration of a record structure

Defining a record type consists of specifying the different fields that make it up. Each field is defined by its type and its name. Declaring a record type is done as follows:

```
TYPE ‹name_of_type› = record
        begin
        field_1 : type_1;
        field_2 : type_2;
          ...
        End;
```

Example

```
TYPE date = record
            Begin
                Day: 1..31 ;
                Month: 1..12 ;
                year: integer;
            End;
VAR D : date ;
```

Note

The fields of a record type can be of any type, whether simple or structured, except for the file type.

3. Accessing Fields in a Record

To access a field in a variable of record type, simply combine the name of the variable with the name of the field, using the full stop as a separator. Example

```
D.Day ← 26 ;
D.Month ←  9;
D.Year ← 2005 ;
```

Note

The record type can be used to define arrays or files that contain structured data.

Application Exercises

1. Given C1 and C2 two complex numbers.
   - Propose a data structure to represent a complex number.
   - Write an algorithm to input C1, C2 and calculate and display C3, the product of C1 and C2.
2. Given D1 and D2 two variables of type Date.
   - Write an algorithm that enters two dates D1, D2, and determines whether date D1 is strictly anterior to (before) date D2.