In this chapter, we introduce a number of applications developed in FORTRAN. The methodology we follow to develop these applications will be shown as we consider each application in detail.

Sorting and Searching are two applications discussed in this chapter. When sorting, we sort (order) elements of a list in either an increasing or a decreasing order. Searching, on the other hand, is the process of finding an element within a list.

9.1 Sorting

Sorting is the process of ordering the elements of any list either in increasing (or ascending) or decreasing (or descending) order. Here, we discuss a method for sorting a list of elements (values) into order according to their arithmetic values. It is also possible to sort elements that have character values since each character has a certain arithmetic value for its representation. This will be discussed in details in Chapter 10.

Sorting in increasing order means that the smallest element in value should be first in the list. Then comes the next smallest element, followed by the next smallest and so on. Figure 1 shows three lists: unsorted (unordered) list, the list sorted in increasing order, and the same list sorted in decreasing order. The exact reverse happens in sorting a list in decreasing order. In the literature, one can find a number of well established techniques for achieving this goal (sorting). Techniques such as insertion sort, bubble sort, quick sort, selection sort, etc. differ in their complexity and speed. In the following section, we introduce a simple sorting technique and its FORTRAN implementation.

<table>
<thead>
<tr>
<th>Unsorted</th>
<th>Increasing order</th>
<th>Decreasing order</th>
</tr>
</thead>
<tbody>
<tr>
<td>73</td>
<td>18</td>
<td>89</td>
</tr>
<tr>
<td>65</td>
<td>40</td>
<td>73</td>
</tr>
<tr>
<td>52</td>
<td>52</td>
<td>65</td>
</tr>
<tr>
<td>18</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>89</td>
<td>65</td>
<td>52</td>
</tr>
<tr>
<td>65</td>
<td>73</td>
<td>40</td>
</tr>
<tr>
<td>40</td>
<td>89</td>
<td>18</td>
</tr>
</tbody>
</table>

Figure 1: Unsorted and sorted lists
9.1.1 A Simple Sorting Technique

The idea of this sorting technique is to select the minimum (or the maximum depending on whether the sorting is in increasing or decreasing order) value within the list and assign it to be the first element of the list. Next, we take the remaining elements and select the minimum among them and assign it to be the second element. This process is repeated until the end of the list is reached. To select the minimum within a list of elements, one has to compare all the elements and keep the minimum value updated.

In the following subroutine, this sorting technique is implemented. Two loops are used in this procedure. The first moves through the elements of the array one after the other and stops at the element before the last element in the array. For each of these elements comparisons are conducted between that element and the rest of the array. So, the second loop moves over the rest of the array elements starting at the element next to the one being considered in the first loop. For example, if the first loop is at element number 3, the second loop would move over the elements from 4 to the last. Within the second loop, element 3 is compared with all the remaining elements starting from the fourth element to the last to make sure that element 3 is less than all of them. If element 5, for example, was found to be less than element 3, we swap the two elements. As we move ahead with the first loop, we are sure that the element we leave is the smallest among the elements that follow it. The FORTRAN subroutine that implements this sorting technique is as follows:

```
SUBROUTINE SORT (A, N)
INTEGER N, A(N), TEMP, K, L
DO 11 K = 1, N - 1
   DO 22 L = K+1, N
      IF (A(K).GT.A(L)) THEN
         TEMP = A(K)
         A(K) = A(L)
         A(L) = TEMP
      ENDIF
22 CONTINUE
11 CONTINUE
RETURN
END
```

Let us now run the above subroutine when the value of N is 5 and the array A consists of the following:

\[
\begin{array}{c}
3 \\
-2 \\
4 \\
9 \\
0 \\
\end{array}
\]

After the first pass (the first iteration of the K-loop), the list becomes:

\[
\begin{array}{c}
-2 \\
3 \\
4 \\
9 \\
0 \\
\end{array}
\]

After the second iteration of the K-loop, the list becomes:

\[
\begin{array}{c}
-2 \\
0 \\
4 \\
9 \\
3 \\
\end{array}
\]

Notice that the 0, the smallest within the 4 remaining elements is the one swapped to the second position. After the third iteration of the K-loop, the list becomes:

\[
\begin{array}{c}
-2 \\
0 \\
3 \\
9 \\
4 \\
\end{array}
\]

After the fourth iteration of the K-loop, the list becomes:
9.2 Searching

As part of any system, information or data might need to be stored in some kind of data structure. One example is one-dimensional arrays. Assume that information about students in some university is stored. Assume again that the IDs of students registered in the current semester are stored in an array STUID. Suppose that an instructor asks the registrar to check whether a student, who has an 882345 as his ID, is registered this semester or not. For the registrar to conduct this check, he has to search within the array STUID for the student who has the ID 882345.

A number of search techniques are well known in computer science. These techniques locate a value within a set of values stored in some data structure. A simple searching technique, namely sequential search, is introduced in the next section.

9.2.1 Sequential Search

Sequential search starts at the beginning of a list (array) and looks at each element sequentially to see if it is the one being searched. This process continues until either the element is found or the list ends, that is all the elements in the list have been checked.

The FORTRAN function that implements this algorithm follows. The function SEARCH searches for the element K in the array A of size N. If the element is found, the index of the element is returned. Otherwise, a zero value is returned.

```
INTEGER FUNCTION SEARCH(A, N, K)
INTEGER N, A(N), K, J
LOGICAL FOUND
SEARCH = 0
J = 1
FOUND = .FALSE.
10 IF (.NOT. FOUND .AND. J .LE. N) THEN
   IF (A(J) .EQ. K) THEN
      FOUND = .TRUE.
      SEARCH = J
   ELSE
      J = J + 1
   ENDIF
   GOTO 10
ENDIF
RETURN
END
```

When the element K is found, the function returns with the position of K. Otherwise, after all the elements have been checked, the function returns with the value zero.

9.3 An Application: Maintaining student grades

Question: Write a program that reads IDs of students together with their grades in some exam. The number of students is read first. The input is given such that each line contains the ID of the student and his grade. Assume the following input:

```
7
886767 94
878787 35
898982 82
867878 63
```

After reading the IDs and the grades, the program must allow us to interactively do the following:

1. **SORT** according to ID
2. **SORT** according to GRADES
3. **CHANGE** a GRADE
4. **EXIT** the program

**Solution:**

We will first write a subroutine **MENU** that gives us the various options listed in the problem and also reads an option. The subroutine **MENU** is as follows:

```fortran
SUBROUTINE MENU (OPTION)
INTEGER OPTION
PRINT*, 'GRADES MAINTENANCE SYSTEM'
PRINT*, ' 0. EXIT THIS PROGRAM'
PRINT*, ' 1. SORT ACCORDING TO ID'
PRINT*, ' 2. SORT ACCORDING TO GRADES'
PRINT*, ' 3. CHANGE A GRADE'
PRINT*, ' ENTER YOUR CHOICE:'
READ*, OPTION
RETURN
END
```

We will now rewrite the subroutine **SORT**, since we need to sort one array and also make the corresponding changes to another array. For example, if we are sorting the array of grades, the swapping of elements in this array must be reflected in the array of IDs as well. Otherwise, the grade of one student would correspond to the ID of another. After sorting, we will print the two arrays in the subroutine. The new subroutine **TSORT** is as follows:

```fortran
SUBROUTINE TSORT (A, B, N)
INTEGER N, A(N), B(N), TEMP, J, K, L
DO 11 K = 1, N - 1
  DO 22 L = K+1, N
    IF (A(K).GT.A(L)) THEN
      TEMP = A(K)
      A(K) = A(L)
      A(L) = TEMP
      TEMP = B(K)
      B(K) = B(L)
      B(L) = TEMP
    ENDIF
  CONTINUE
11 CONTINUE
PRINT*, 'SORTED DATA :'
DO 33 J = 1, N
  PRINT*, A(J), B(J)
33 CONTINUE
RETURN
END
```

Note that we are sorting array A but making all the corresponding changes in array B. To this subroutine, we can pass the array of grades as array A and the array of IDs as array B. The subroutine then returns the array of grades sorted but at the same time
makes the corresponding changes to the array of IDs. If to this subroutine, we pass the array of IDs as array A and the array of grades as array B, the subroutine returns the array of IDs sorted but at the same time makes the corresponding changes to the array of grades.

To change a grade, we are given the ID of the student. We need to search the array of IDs for the given ID. We can use the function SEARCH we developed in Section 9.2. We can pass the array of IDs to the dummy array A and the ID to be searched to the dummy argument K. Note that the function SEARCH returns a zero if the ID being searched is not found.

Using the subroutines MENU and TSORT, and the function SEARCH, we develop the main program as follows:

```
INTEGER GRADES(20), ID(20)
INTEGER SEARCH, SID, NGRADE, OPTION, K, N
PRINT*, 'ENTER NUMBER OF STUDENTS'
READ*, N
DO 10 K = 1, N
   PRINT*, 'ENTER ID AND GRADE OF STUDENT ', K
   READ*, ID(K), GRADES(K)
10 CONTINUE
CALL MENU (OPTION)
15 IF (OPTION .NE. 0) THEN
   IF (OPTION .EQ. 1) THEN
      CALL TSORT(ID, GRADES, N)
   ELSEIF (OPTION .EQ. 2) THEN
      CALL TSORT(GRADES, ID, N)
   ELSEIF (OPTION .EQ. 3) THEN
      PRINT*, 'ENTER ID & THE NEW GRADE'
      READ*, SID, NGRADE
      K = SEARCH(ID, N, SID)
      IF (K.NE.0) THEN
         GRADES(K) = NGRADE
      ELSE
         PRINT*, 'ID : ' ,SID, ' NOT FOUND'
      ENDIF
   ELSE
      PRINT*, 'INPUT ERROR '
   ENDIF
   CALL MENU (OPTION)
   GOTO 15
ENDIF
END
```

The main program first reads the two arrays ID and GRADES each of size N. Then it displays the menu and reads an option from the screen into the variable OPTION using subroutine MENU. If the input option is 1, the subroutine TSORT is called in order to sort IDs. If the input option is 2, the subroutine TSORT is called in order to sort the grades. If the input option is 3, the ID to be searched (SID) and the new grade (NGRADE) are read, and the function SEARCH is invoked. If the ID is found, the corresponding grade in array GRADES is changed. Otherwise, a message indicating that the SID is not found is printed. The main program runs until option 4 is chosen.

### 9.4 Exercises

1. Modify the application given in Section 9.3 as follows:
a. Add an option that will list the grade of a student given his ID.
b. Given a grade, list all IDs who scored more than the given grade.
c. Add an option to find the average of all the grades.
d. Add an option to find the maximum grade and the corresponding ID.
e. Add an option to find the minimum grade and the corresponding ID.
f. Add an option to list the IDs of all students above average.

2. The seating arrangement of a flight is stored in a data file FLIGHT containing six lines. Each line contains three integers. A value of 1 represents a reserved seat, and a value of 0 represents an empty seat. The contents of flight are:

```
1 0 1
0 1 1
1 0 0
1 1 1
0 0 1
0 0 0
```

Write an interactive program which has a menu with the following options:

0. Exit
1. Show number of empty seats
2. Show Empty seats
3. Reserve a seat
4. Cancel a seat

The program first reads from the data file FLIGHT and stores the data in a two-dimensional integer array seats of size $6 \times 3$ row-wise. Then:

a. If option 1 is chosen, the main program passes the array seats to an integer function NEMPTY which returns the number of empty seats. Then the main program prints this number.

b. If option 2 is chosen, the main program passes the array seats to a subroutine ESEATS which returns the number of empty seats and the positions of all empty seats in a two-dimensional integer array EMPTY of size $18 \times 2$. Then, the main program prints the array EMPTY row-wise.

c. If option 3 is chosen, the user is prompted to enter the row number and the column number of the seat to be reserved. The main program then passes these two integers together with the array SEATS to a logical function RESERV which reserves a seat if it is empty and returns the value .true. to the main program. If the requested seat is already reserved or if the row or column number is out of range the function returns the value .false. to the main program. The main program then prints the message SEAT RESERVED or SEAT NOT AVAILABLE respectively.

d. If option 4 is chosen, the user is prompted to enter the row number and the column number of the seat to be canceled. The main program then passes these two integers together with the array SEATS to a logical function CANCEL which cancels a seat if it is reserved and returns the value .true. to the main program. If the requested seat is already empty or if the row or column number is out of range the function returns the
value .false. to the main program. The main program then prints the message SEAT CANCELED or WRONG CANCELLATION respectively.

e. If option 0 is chosen, the main program stops immediately if no changes were made to the array seats. otherwise, the main program closes the data file flight and then opens it to write into the data file the new seating arrangement stored in the array seats before stopping.

9.5 Solutions to Exercises

1. For each of the following subprograms, appropriate changes must be made to the subroutine MENU on page 190 and the main program on page 192.

a. SUBROUTINE LISTGR(ID, GRADES, N)
   INTEGER N, GRADES(N), ID(N), SID, SEARCH, K
   PRINT*, 'ENTER STUDENT ID'
   READ*, SID
   C USING SEARCH FUNCTION ON PAGE 189
   K = SEARCH(ID, N, SID)
   IF (K .NE. 0) THEN
      PRINT*, 'GRADE OF ID #', SID, ' IS ', GRADE(K)
   ELSE
      PRINT*, 'ID #', SID, ' DOES NOT EXIST'
   ENDIF
   RETURN
END

b. SUBROUTINE LISALL(ID, GRADES, N)
   INTEGER N, GRADES(N), ID(N), SGR, SEARCH, K
   PRINT*, 'ENTER STUDENT GRADE'
   READ*, SGR
   PRINT*, 'ID OF STUDENTS WITH GRADE = ', SGR
   DO 10 K = 1, N
      IF (GRADE(K) .GE. SGR) PRINT*, ID(K)
   10 CONTINUE
   RETURN
END

c. REAL FUNCTION AVERAG(GRADES, N)
   INTEGER N, GRADES(N), K
   REAL SUM
   SUM = 0
   DO 10 K = 1, N
      SUM = SUM + GRADE(K)
   10 CONTINUE
   AVERAG = SUM / N
   RETURN
END
d.  

```fortran
SUBROUTINE LISMAX(ID, GRADES, N)
INTEGER N, GRADES(N), ID(N), INDEX, MAXGRD, K
INDEX = 1
MAXGRD = GRADES(1)
DO 10 K = 1, N
   IF( GRADES(K) .GT. MAXGRD) THEN
      MAXGRD = GRADES(K)
      INDEX  = K
   ENDIF
10 CONTINUE
PRINT*, 'MAXIMUM GRADE = ', MAXGRD
PRINT*, 'ID OF STUDENT WITH MAXIMUM GRADE = ', ID(INDEX)
RETURN
END
```

e.  

```fortran
SUBROUTINE LISMIN(ID, GRADES, N)
INTEGER N, GRADES(N), ID(N), INDEX, MINGRD, K
INDEX = 1
MINGRD = GRADES(1)
DO 10 K = 1, N
   IF( GRADES(K) .LT. MINGRD) THEN
      MINGRD = GRADES(K)
      INDEX  = K
   ENDIF
10 CONTINUE
PRINT*, 'MINIMUM GRADE = ', MINGRD
PRINT*, 'ID OF STUDENT WITH MINIMUM GRADE = ', ID(INDEX)
RETURN
END
```

f.  

```fortran
SUBROUTINE LISIDS(ID, GRADES, N)
INTEGER N, GRADES(N), ID(N), K
REAL AVERAG, AVG
C USING AVERAGE FUNCTION IN PART C
AVG = AVERAG(GRADES, N)
PRINT*, 'ID OF STUDENTS ABOVE AVERAGE'
DO 10 K = 1, N
   IF( GRADE(K) .GT. AVG) PRINT*, ID(K)
10 CONTINUE
RETURN
END
```
Ans 2.

```fortran
INTEGER SEATS(6,3), EMPTY(18,2), NEMPTY, OPTION, ROW, CLMN
INTEGER J, K
LOGICAL RESERV, CANCEL, CHANGE
OPEN (UNIT=40, FILE = 'FLIGHT', STATUS = 'OLD')
DO 10 J = 1, 6
   READ (40,*)(SEATS(J,K), K=1,3)
10 CONTINUE
CHANGE = .FALSE.
CALL MENU(OPTION)
15 IF(OPTION .NE. 0) THEN
   IF(OPTION .EQ. 1) THEN
      PRINT*,'THE NUMBER OF EMPTY SEATS = ', NEMPTY(SEATS)
   ELSEIF(OPTION .EQ. 2) THEN
      CALL ESEATS(SEATS, EMPTY, N)
      PRINT*,'EMPTY SEATS:'
      DO 20 J = 1, N
         PRINT*,(EMPTY(J,K), K=1,2)
      20 CONTINUE
   ELSEIF(OPTION .EQ. 3) THEN
      PRINT*,'ENTER NEEDED SEATS ROW AND COLUMN NUMBER'
      READ*,ROW, CLMN
      IF(RESERV(SEATS, ROW, CLMN)) THEN
         PRINT*,'SEAT RESERVED'
         CHANGE = .TRUE.
      ELSE
         PRINT*,'SEAT NOT AVAILABLE'
      ENDIF
   ELSEIF(OPTION .EQ. 4) THEN
      PRINT*,'ENTER ROW# AND COLUMN# OF THE SEAT TO CANCEL'
      READ*,ROW, CLMN
      IF(CANCEL(SEATS, ROW, CLMN)) THEN
         PRINT*,'SEAT CANCELED'
         CHANGE = .TRUE.
      ELSE
         PRINT*,'WRONG CANCELLATION'
      ENDIF
   ELSE
      PRINT*,'WRONG OPTION'
   ENDIF
15 ENDIF
IF(CHANGE) THEN
   CLOSE(40)
   OPEN (UNIT=40, FILE = 'FLIGHT', STATUS = 'OLD')
   DO 25 J = 1, 6
      WRITE (40,*) (SEATS(J,K), K=1,3)
25 CONTINUE
ENDIF
END
```
SUBROUTINE MENU(OPTION)
INTEGER OPTION
PRINT*, '***** FLIGHT RESERVATION *****'
PRINT*, '1. NUMBER OF EMPTY SEATS'
PRINT*, '2. EMPTY SEATS'
PRINT*, '3. RESERVE SEAT'
PRINT*, '4. CANCEL SEAT'
PRINT*, '5. EXIT'
PRINT*, 'ENTER YOUR OPTION:'
READ*, OPTION
RETURN
END

INTEGER FUNCTION NEMPTY(SEATS)
INTEGER SEATS(6,3), J, K
NEMPTY = 0
DO 30 J = 1, 6
   DO 35 K = 1, 3
      IF(SEATS(J,K) .EQ. 0 )THEN
         NEMPTY = NEMPTY + 1
      ENDIF
   CONTINUE
35 CONTINUE
30 CONTINUE
RETURN
END

SUBROUTINE ESEATS(SEATS, EMPTY, N)
INTEGER N, SEATS(6,3), EMPTY(18,2), J, K
N = 1
DO 40 J = 1, 6
   DO 45 K = 1, 3
      IF(SEATS(J,K) .EQ. 0 )THEN
         EMPTY(N,1) = J EMPTY(N,2) = K
         N = N + 1
      ENDIF
   CONTINUE
45 CONTINUE
40 CONTINUE
N = N - 1
RETURN
END

LOGICAL FUNCTION RESERV(SEATS, ROW, CLMN)
INTEGER SEATS(6,3), ROW, CLMN
RESERV = .FALSE.
IF(ROW .GE. 1 .AND. ROW .LE. 6)THEN
   IF(CLMN .GE. 1 .AND. CLMN .LE. 3)THEN
      IF(SEATS(ROW,CLMN) .EQ. 0 )THEN
         SEATS(ROW,CLMN) = 1
         RESERV = .TRUE.
      ENDIF
   ENDIF
ENDIF
ENDF
END
RETURN
END
LOGICAL FUNCTION CANCEL(SEATS, ROW, CLMN)
INTEGER SEATS(6,3), ROW, CLMN
CANCEL = .FALSE.
IF(ROW .GE. 1 .AND. ROW .LE. 6) THEN
   IF(CLMN .GE. 1 .AND. CLMN .LE. 3) THEN
      IF(SEATS(ROW,CLMN) .EQ. 1) THEN
         SEATS(ROW,CLMN) = 0
         CANCEL = .TRUE.
      ENDIF
   ENDIF
ENDIF
RETURN
END