Data Type Classification:

The simple data types are:

(a) Integers

(b) Real Numbers

(c) Characters

(d) Booleans

Integers are simple numbers that do not contain fractional parts such as 5, -52, and 3,422.

Real numbers are numbers that include fractional parts such as 2.5, 3.667, -5.0, .00034.

Characters: All of the symbols that can be produced by pressing keys on a keyboard. Characters are used to generate human readable text such as English sentences. The Data Type **Char** is used to represent characters.

Booleans: The values false and true.

Primitive Data Types

Every computer language supports a set of native data types.

These types may be native (built-in) either to the machine on which the programs are run or to the compiler or interpreter that is translating these programs.

Fixed-Point Representation of Real Numbers

The usual way of representing **real** numbers is to write the number with the decimal point fixed in its correct position between the two appropriate digits,

e.g., 13.75 or 3862.4. This is very useful in data processing for examp sums of money are to be processed or printed.

However, this representation becomes laborous and cumbersome when dealing with several large or very small numbers e.g. 1375000000, 386240000, 0.000001375, 0.0000018. The answer is to use the floating- point representation.

Floating-Point Notation

There are many variations of floating-point notation, each with its own individual characteristics.

The key concept is that a real number is represented by a number, called a MANTISSA, times a BASE raised to an integer power, called an EXPONENT. The base is usually fixed and the mantissa and exponent vary to represent different real numbers.

For example, if the base is fixed at 10, the number 387.53 could be represented as 38,753 times 10 to the -2 power. The mantissa is 38753 and the exponent is -2.

Other possible representations are $.38753 \times 10^3$ and $387.53 * 10^0$

The floating-point representation of 640,000 is $6.4 * 10^5$. The mantissa is

6.4, the exponent 5 shows the power of 10 to which 6.4 is raised.

SCIENTIFIC NOTATION

The scientific notation is a floating-point method of representing a number, especially a very large number or very small one, in which numbers are expressed as products consisting of a number between 1 and 10 multiplied by a power of 10.

Scientific notation commonly uses the letter E in place of "times 10", as in 5.0E3, meaning 5.0 times 10 to the third power, or 10^3.

Thus 640,000 = 6.4E+05 or 6.4E5.

Further Examples:

Fixed Point Representation	Floating -Point Representation
13.75	$1.375 * 10^{1}$
137.5	$1.375 * 10^{2}$
1375.	$1.375 * 10^{3}$
1375000000.	$1.375 * 10^{9}$
1.375	$1.375 * 10^{0}$
0.1375	$1.375 * 10^{-1}$
0.01375	$1.375 * 10^{-2}$
0.001375	$1.375 * 10^{-3}$
3862.4	$3.8624 * 10^{3}$
386240000.	$3.8624 * 10^{8}$
.00000038624	$3.8624 * 10^{-7}$

For $1.375 * 10^{3}$

1.375 is the Mantissa

10 is the base or radix

3 is the exponent

CMP 312 Jacobs Copy

FUNDAMENTALS OF DATA STRUCTURES-4

ARRAYS:

We can define array as follows.

- An array is a group of objects referenced with the same variable name. The individual values in an array are elements. Array elements are also variables. You "dimension" an array when you use it the first time or line Each element in array is referred to by an array variable subscripted with an integer or an integer expression.

- An array is a collection of variables of the same type that are referred to through a common name.

- An array A is a set of items, of the same type, which are so arranged that an ordered set of integers uniquely defines the position of each item of the array and provides a method of accessing each item directly. Each individual integer in the ordered set is called a subscript.

Dimension Of an Array: If the number of elements in the ordered set of subscripts is N, then the array is said to be of dimension N. So when there is only one subscript, the array is said to be one-dimensional, When there are two subscripts, the array is said to be two-dimensional when there are three subscripts, the array is said to be three-dimensional. Generally, when there are more than ONE subscripts, the array is said to be a multi-dimensional array.

In programming languages, all arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element. Arrays may have from one to several dimensions.

Arrays store collections of data. Array is the most common data structure

used to store collections of elements. Arrays are convenient to declare and

provide the easy syntax to access any element by its index number. Once

the array is set up, access to any element is convenient and fast.

In programming, like other variables, arrays must be explicitly ***declared*** so that the compiler may allocate space for them in memory.

In computer science, the obvious way to store an ordered collection of items is as an array.

Array items are typically stored in a sequence of contiguous computer memory locations, but to discuss them, we need a convenient way to write them down on paper. We can just write the items in order, separated by commas and enclosed by square brackets. Thus,

[1; 4; 17; 3; 90; 79; 4; 6; 81]

is an example of an array of integers. If we call this array **a**, we can write it as:
a = [1; 4; 17; 3; 90; 79; 4; 6; 81]

Size of an Array:
This array a has 9 items, and hence we say that its ***size*** is 9. In everyday life, we usually start counting from 1. When we work with arrays in computer science, however, we more often (though not always) start from 0 (in C and C++). Thus, for our array a, its positions are 0; 1; 2; ; ; 7; 8. The element in the 8th position is 81, and we use the notation a[8] to denote this element. More generally, for any integer i denoting a position, we write a[i] to denote the element in the ith position. This position i is called an index and the plural is indices). Then, in the above example,
a[0] = 1,
a[1] = 4,
a[2] = 17, and so on.

Arrays may have from one to several dimensions.

One-dimensional Array:

Note:

Declaring an Array:

In any program arrays must be properly and correctly declared before use.

Format:

data-type Array-name[Array-Size];

e.g. (C/C++)

int Array1[5]; one-dimensional

int Array2[10] [5]; two-dimensional

Usually the size of a dimension is normally given during the declaration of an array. It means that you have to count each element in an array to determine the size. It could be tedious to do so, though, especially if there are many elements in an array.

The good news is that the C/C++ compiler can actually calculate a dimension size of an array automatically if an array is declared as an unsized array. For example, when the compiler sees the following unsized array:

int grade[ ] = { 20, 30, 40, 50, 60, 70, 80, 90, 100};

it will create an array big enough to store all the elements.

Likewise, you can declare a multidimensional unsizedarray. However, you have to specify all but the leftmost (thatis, the first) dimension size. For instance, the compiler can reserve enough memory space to hold all elements in the following two-dimensional unsized array:

Example 1:

```
char list5[ ] [2] = {
'a', 'A',
'b', 'B',
'c', 'C',
'd', 'D',
'e', 'E',
'f', 'F',
'g', 'G' };
```

Example 2:

```
int grade[ ] [2] = {
30, 75,
45, 60,
75, 55,
40, 50,
60, 80};
```

++++++++++++++++++++==========

Example: A programming class has 5 students. Each programming assignment test is graded on a scale of 0 to 100. The grades earned by the class in a single assignment can be arranged in the form of an array called GRADE1.

GRADE1:

| 92 |

| 75 |

| 60 |

| 35 |

| 70 |

Above,

GRADE1 (2) = 75

GRADE1 (5) = 70

Two-dimensional Arrays:

If in the programming class above there are 4 programming tests and each graded on a scale of 0 to 100,the layout of the grades may be as follows:

TESTS Students	1.	2.	3.	4.
1.	92	75	82	55
2.	75	60	70	45
3.	60	17	50	40
4.	35	80	65	44
5.	70	75	77	51

Let this array be named GRADE2. This s an example of a 2-dimensional array. The subscripts are (student, test).

Thus GRADE2(1,1) = 92

GRADE2(3,3) = 50

GRADE2(4,2) = 80

and GRADE2(2,4) = 45

Generally, GRADE2(I,J) is the grade of the I^{th} student in the J^{th} Test.

A two-dimensional array is a collection of components of the same type that is structured in two dimensions. Individual components are accessed

by their position within each dimension.

Three types are associated with a two-dimensional array data type: the type of the items to be stored in the individual places in the structure, the type of the index for the first dimension, and the type of the index for the second dimension.

In C++ the type of both dimensions must be integral.

Example:

```
const int MAX_ROWS = 10;

const int MAX_COLUMS = 5;

typedef float ItemType;

ItemType twoDimAry [MAX_ROWS] [MAX_COLUMNS];
```

twoDimAry is an array variable that has 10 rows and 5 columns. Each row and column entry is of type float.

The following short code sets all the entries in twoDimAry to zero.

```
for (int column = 0; column < MAX_COLUMNS; column++)

    for ( int row = 0;  row < MAX_ROWS; row++ )

        twoDimAry [row] [column] = 0.0;
```

Table Processing

Just as a one-dimensional array data type is the structure used to represent items in a list, a two-dimensional array data type is the structure that is often used to represent items in a table. The number of rows and columns in the two-dimensional array is fixed at compile time. The number of rows and columns in the table can vary as the program executes. Therefore, each dimension should have a length parameter associated with it that contains the number of rows or columns actually used.

Example:

A chain of 8 stores, each having 4 depart nents, may list its weekly sales as in FIG. 1 .

Such data can be stored in the computer using a two-dimensional array in which the first subscript denotes the store and the second subscript denotes the department. If SALES is the name given to the array, then

SALES[1,1] = 2870.

SALES [1,2] = 805.

SALES [1,3] = 3210

SALES[4,4] = 1250

SALES [5,3] = 8300

...

SALES [8,4] = 9820

STORES	DEPARTMENTS 1	2	3	4
1	2870	805	3210	1260
2	2196	1223	2525	1744
3	3250	1070	3686	3955
4	6745	9850	1055	1250
5	20555	7500	8300	8000
6	15750	4500	3750	7350

	5500	11600	8500	4150
s	11900	45800	5000	4120

FIG. 1

The size of this array is denoted by 8 X 4 (read 8 by 4), since it contains 8 rows (the horizontal lines of numbers) and 4 columns (the vertical lines of numbers).

One -dimensional arrays are, by far, the most common array type used in most programming languages. However, there are many situations in which it is natural to use a two-dimensional array. These include:

- storing and manipulating matrices;
- processing data which can be arranged in rows and columns;
- board games, where the board has a rectangular layout, for example chess, checkers (draughts) or tic-tac-toe.

Three-Dimensional Array:

An Example:

Following from the above example, suppose now we have Tests for *levels* of students.

Let us call this array GRADE3. Therefore we will have 3 subscripts.

Thus GRADE3(I,J,K) is the grade of the I^{th} student in the J^{th} Test, the level of the student is K.

By varying values of the subscripts any particular element/entry in the array can be accessed. In the examples above the elements are all integers, that is, the arrays are arrays of integers, but we can have also arrays of characters, arrays of boolean, etc.