**Date: 14-04-2021**

**Branch: CSE and EE/3 year**

**Topic:** Basic computer data types,

Complement of Number and

Fixed point of representation of number

**Time: 08:10 AM -9:05 AM**

Let's discuss about a very simple but very important concept available in almost all the programming languages which is called **data types**. As its name indicates, a data type represents a type of the data which you can process using your computer program. It can be numeric, alphanumeric, decimal, etc.

Let’s keep Computer Programming aside for a while and take an easy example of adding two whole numbers 10 & 20, which can be done simply as follows −

10 + 20

Let's take another problem where we want to add two decimal numbers 10.50 & 20.50, which will be written as follows −

10.50 + 20.50

The two examples are straightforward. Now let's take another example where we want to record student information in a notebook. Here we would like to record the following information −

Name:

Class:

Section:

Age:

Sex:

Now, let's put one student record as per the given requirement −

Name: Zara Ali

Class: 6th

Section: J

Age: 13

Sex: F

The first example dealt with whole numbers, the second example added two decimal numbers, whereas the third example is dealing with a mix of different data. Let's put it as follows −

* Student name "Zara Ali" is a sequence of characters which is also called a string.
* Student class "6th" has been represented by a mix of whole number and a string of two characters. Such a mix is called alphanumeric.
* Student section has been represented by a single character which is 'J'.
* Student age has been represented by a whole number which is 13.
* Student sex has been represented by a single character which is 'F'.

This way, we realized that in our day-to-day life, we deal with different types of data such as strings, characters, whole numbers (integers), and decimal numbers (floating point numbers).

Similarly, when we write a computer program to process different types of data, we need to specify its type clearly; otherwise the computer does not understand how different operations can be performed on that given data. Different programming languages use different keywords to specify different data types. For example, C and Java programming languages use **int** to specify integer data, whereas **char** specifies a character data type.

Subsequent chapters will show you how to use different data types in different situations. For now, let's check the important data types available in C, Java, and Python and the keywords we will use to specify those data types.

The computer stores data in different ***formats*** or ***types***. The number ***10*** can be stored as numeric value as in *"10 dollars"* or as character as in the address *"10 Main Street"*.  So how can the computer tell? Once again the computer doesn't care, it is your responsibility to ensure that you get the correct data out of it. (For illustration character 10 and numeric 10 are represented by 0011-0001-0011-0000 and 0000-1010 respectively — you can see how different they are.) Different programming launguages have different ***data types***, although the foundamental ones are usually very similar.

### C++ Basic Data Types *(C++ specific)*

C++ has many data types. The followings are some basic data types you will be facing in these chapters. Note that there are more complicated data types. You can even create your own data types. Some of these will be discussed later in the tutorial.

|  |  |  |  |
| --- | --- | --- | --- |
| *Data Type* | *Bytes* | *Data Range* | *Remarks* |
| char | 1 | ASCII -128 to127 |  |
| unsigned char | 1 | ASCII 0 to 255 | including high ASCII chars |
| int | 2 | -32768 to 32767 | Integer |
| unsigned (unsigned int) | 2 | 0 to 65535 | non-negative integer |
| long int | 4 | ± 2 billions | double sized integer |
| unsigned long int | 4 | 0 to 4 billion | non-negative long integer |
| float | 4 | 3.4 ±e38 | 6 significant digits |
| double | 8 | 1.7 ±e308 | 15 significant digits |

Python Data Types

Python has five standard data types but this programming language does not make use of any keyword to specify a particular data type, rather Python is intelligent enough to understand a given data type automatically.

* Numbers
* String
* List
* Tuple
* Dictionary

Here, Number specifies all types of numbers including decimal numbers and string represents a sequence of characters with a length of 1 or more characters. For now, let's proceed with these two data types and skip List, Tuple, and Dictionary, which are advanced data types in Python.

# Complement Arithmetic

5 - 3= 5 (+3) =5 + 2s(+3)

Complements are used in the digital computers in order to simplify the subtraction operation and for the logical manipulations. For each radix-r system (radix r represents base of number system) there are two types of complements.

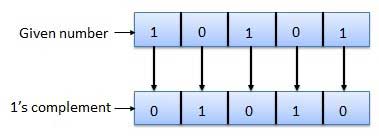
|  |  |  |
| --- | --- | --- |
| **S.N.** | **Complement** | **Description** |
| 1 | Radix Complement | The radix complement is referred to as the r's complement |
| 2 | Diminished Radix Complement | The diminished radix complement is referred to as the (r-1)'s complement |

## Binary system complements

As the binary system has base r = 2. So the two types of complements for the binary system are 2's complement and 1's complement.

### 1's complement

The 1's complement of a number is found by changing all 1's to 0's and all 0's to 1's. This is called as taking complement or 1's complement. Example of 1's Complement is as follows.

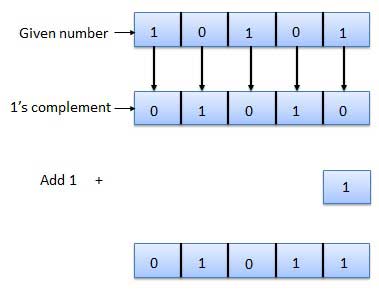


### 2's complement

The 2's complement of binary number is obtained by adding 1 to the Least Significant Bit (LSB) of 1's complement of the number.

2's complement = 1's complement + 1

Example of 2's Complement is as follows.



10-6= 10 + (-6)=4

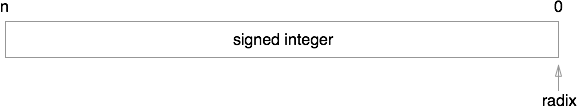
10=1010

6= 0110 -6= 1001+1=1010

1010+1010= 0100

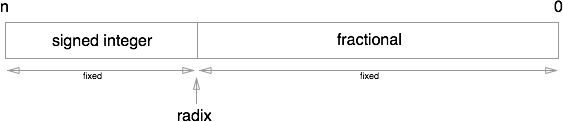
## Fixed Point Representation

**Fixed Point Notation** is a representation of our fractional number as it is stored in memory. In Fixed Point Notation, the number is stored as a signed integer in [two’s complement format](https://andybargh.com/signed-numbers-in-binary/).



12.25

On top of this, we apply a notional split, locating the radix point (the separator between integer and fractional parts) a fixed number of bits to the left of its notational starting position to the right of the least significant bit. I’ve illustrated this in the diagram below.



10.56=105.6\*10^1

When we interpret the bits of the signed integer stored in memory we reposition the radix point by multiplying the stored integer by a fixed scaling factor. The scaling factor in binary is always 2 raised to a fixed exponent. As the scaling factor is a power of 2 it relocates the radix point some number of places to the left or right of its starting position.

During this conversion there are three directions that the radix point can be moved:

* **The radix point is moved to the right:** This is represented by a scaling factor whose exponent is 1 or more. In this case additional zeros are appended to the right of the least-significant bit and means that the actual number being represented is larger than the binary integer that was stored.
* **The radix point remains where it is:** This is represented by a scaling factor whose exponent is 0 and means that the integer value stored is exactly the same as the integer value being represented.
* **The radix point is moved to the left:** This is represented by a scaling factor whose exponent is negative. This means that the number being represented is smaller than the integer number that was stored and means that the number being represented has a fractional component

**112.**56=11.256\*10^-1

Let’s take a look at a couple of examples.

### Examples of Fixed Point Numbers

Lets assume we have an 8-bit signed binary number 000110112 that is stored in memory using 8-bits of storage (hence the leading zeros).

In our first scenario, lets also assume this number was stored as a signed fixed-point representation with a scale factor of 22.

As our scale factor is greater than 1, when we translated the bits stored in memory into the number we are actually representing, we move the radix point two places to the right. This gives us the number: 11011002 (Note the additional zeros that are appended to the right of the least significant bit).

In our second scenario, let us assume that we start off with the same binary number in memory but this time we’ll assume that it is stored as a signed fixed-point representation with a scale factor of 2-3. As the exponent is negative we move the radix point three places to the left. This gives us the number 00011.0112

### Advantages and Disadvantages of Fixed Point Representation

The major advantage of using a fixed-point representation is performance. As the value stored in memory is an integer the CPU can take advantage of many of the optimizations that modern computers have to perform integer arithmetic without having to rely on additional hardware or software logic. This in turn can lead to increases in performance and when writing your apps, can therefore lead to an improved experience for your users.

However, there is a downside! Fixed Point Representations have a relatively limited range of values that they can represent.

So how do we work out the maximum and minimum numbers that can be stored in a fixed-point representation and determine whether it is suitable for our needs? All we do is take the largest and smallest integer values that can be stored in the given number of bits and multiply that by the scale factor associated with our fixed-point representation. For a given signed binary number using b bits of storage with a scale factor of f the maximum and minimum values that can be stored are:

Minimum: −2b−1/2f

Maximum: (2b−1−1)/2f

If the number you want to represent fits into this range then things are great. If it doesn’t though, you have to look for an alternative! This is where Floating Point Notation comes in.