CS5600 Computer Systems

CHAR DATA TYPE - Its not a bug, It is a feature

Name : Jayaraman Venkatesan

1 Some background

DataType in programming languages are assigned bits to store them in memory. They are divided into signed and unsigned among themselves to accommodate signed values. By default if we declare the following code in any machine across the world :

```
int a;
```

The compiler will allocate a memory in stack to accommodate a integer value. Additionally to identify whether the number is positive or negative, a "signed bit" is allocated by default. However if we do not want the compiler to waste bits on signs and instead use those bits to store large number we can "explicitly" define it as "unsigned".

2 Char Datatype is different !

2.1 Part 1 (Theory) :

In C, char datatype is a special form of "int" data type that encodes ASCII values. Breaking this down, the below code will work as follows :

```
char a = 'A';  \\initializes a as 'A' which has ASCII code of 65
printf("%d",a); \\since we are printing integer portion of a, 65 will be printed!
```

Since it internally works on integer values, char datatype too has "signed" and "unsigned" variants.

But unlike the assumption made in the above section, where in the case of integers the memory will automatically be assigned for signed bits and default "int" is "signed int", the char data type default is not fixed! And depends on the compilers running on different machine architecture!

The signed bits are automatically allocated to handle negative values in x86 systems so they are signed by default, whereas in ARM based linux systems, they are not allocated and unsigned by default! [see references (2)]

2.2 Part 2 (Where will it matter) :

If you are coding in x86 based systems(intel based), the following code will work like this:

```
char a = 'A';      \\  gets 'A' stored which has ASCII value of 65
char b = a - 100; \\ The arithmetic happens in ascii values, and hence 65-100=-35
printf("%d", b);  \\ Prints -35
```

This happens because of x86 systems, this is because the data type is signed by default in its architecture.

The same code behaves like this in ARM based linux :

```
char a = 'A';      \\  gets 'A' stored which has ASCII value of 65
char b = a - 100; \\ The arithmetic happens in ascii values, and hence 65-100=-35
                  \\ since they cant handle negetive numbers its set 221
printf("%d", b);  \\ Prints 221
```

But why 221 ? this is because unsigned char data type ranges from 0 - 255 only! [see references (1)]
if we subtract more than that present, it will be rounded back i.e
b = 65 - 100
b = -35 $\sim 2^8 - 35 = 221$

3 References

1. Datatype sizes
2. C programming by Dennis Ritchie. Pg.36