C Modifiers

Think about the following situation!

- Let X be a variable such that its value varies from 0 to 10.

- In terms of memory storage, what do you think is wrong with the following statement.

```
int X;

scanf(“%d”, &X);
```

Now Think about this!

- Assume a 32-bit computer. You need to store integer from 0 to 2200000000.

- In terms of memory storage, what do you think is wrong with the following statement.

```
int X;

scanf(“%d”, &X);
```

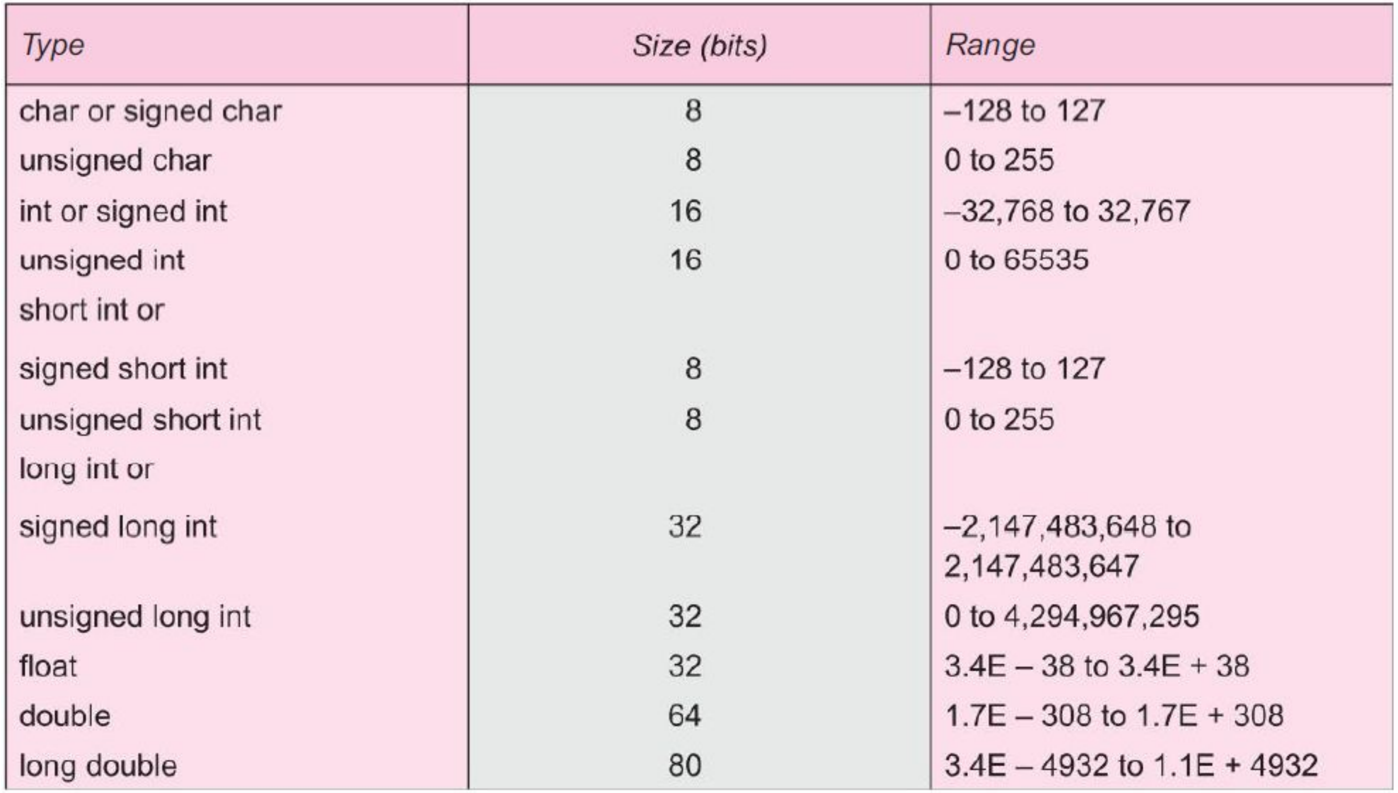
Modifiers in C

- Modifiers are prefixed with basic data types to modify (either increase or decrease) the amount of storage space allocated to a variable.
- For example, storage space for int data type is 4 byte for 32 bit processor. We can increase the range by using long int which is 8 byte. We can decrease the range by using short int which is 2 byte.
- 4 Types
  - Short
  - Long
  - Signed
  - Unsigned

Contd.

- Each of these type modifiers can be applied to the base type int.
- Type modifiers signed and unsigned can also be applied to the base type char.
- In addition, long can be applied to double.

Storage Space



Type	*Size (bits)*	*Range*
char or signed char	8	–128 to 127
unsigned char	8	0 to 255
int or signed int	16	–32,768 to 32,767
unsigned int	16	0 to 65535
short int or		
signed short int	8	–128 to 127
unsigned short int	8	0 to 255
long int or		
signed long int	32	–2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
float	32	3.4E – 38 to 3.4E + 38
double	64	1.7E – 308 to 1.7E + 308
long double	80	3.4E – 4932 to 1.1E + 4932

Short

It limits user to store small integer values from -32768 to 32767. It can be used only on int data type.

Ex.

```
short int a = 10;

printf(“%hd”, a);
```

Long

It allows user to stores very large number (something like 9 Million Trillion) from -9223372036854775808 to 9223372036854775807.

Ex.

```
long int a = 10;

printf(“%ld”, a);
```

Signed

It is default modifier of int and char data type if no modifier is specified. It says that user can store negative and positive values.

Ex.

```
signed int myNegativeIntegerValue = -544;

printf(“%d”, myNegativeIntegerValue );
```

Unsigned

When user intends to store only positive values in the given data type (int and char).

Ex.

```
unsigned int myIntegerValue = 486;

printf(“%u”, myIntegerValue);
```

Type Casting

Consider the following situations

```
int X = 10;

float Y;

Y = X;

printf(“%f”, Y);
```

```
O/P: 10.000000
```

```
float Y = 1 / 2;

printf(“%f”, Y);
```

```
O/P: 0.000000
```

Solution

```
float Y = 1.0 / 2;

printf(“%f”, Y);
```

Type Casting or Type Conversion

- Converting values of one type to other.
  - Ex. int to float and float to int
- Can be for other types as well.
- Two types: implicit and explicit

```
Ex. int x = 5;

float y = x;            //implicit conversion, y gets 5.0

                        //value of x is not changed
```

Example

```
int k = 5;

float y = k / 10;
```

- y will get value 0.0
- Since both k and 10 are integers, k / 10 will result in integer.
- Next 0 will be implicitly converted into 0.0.

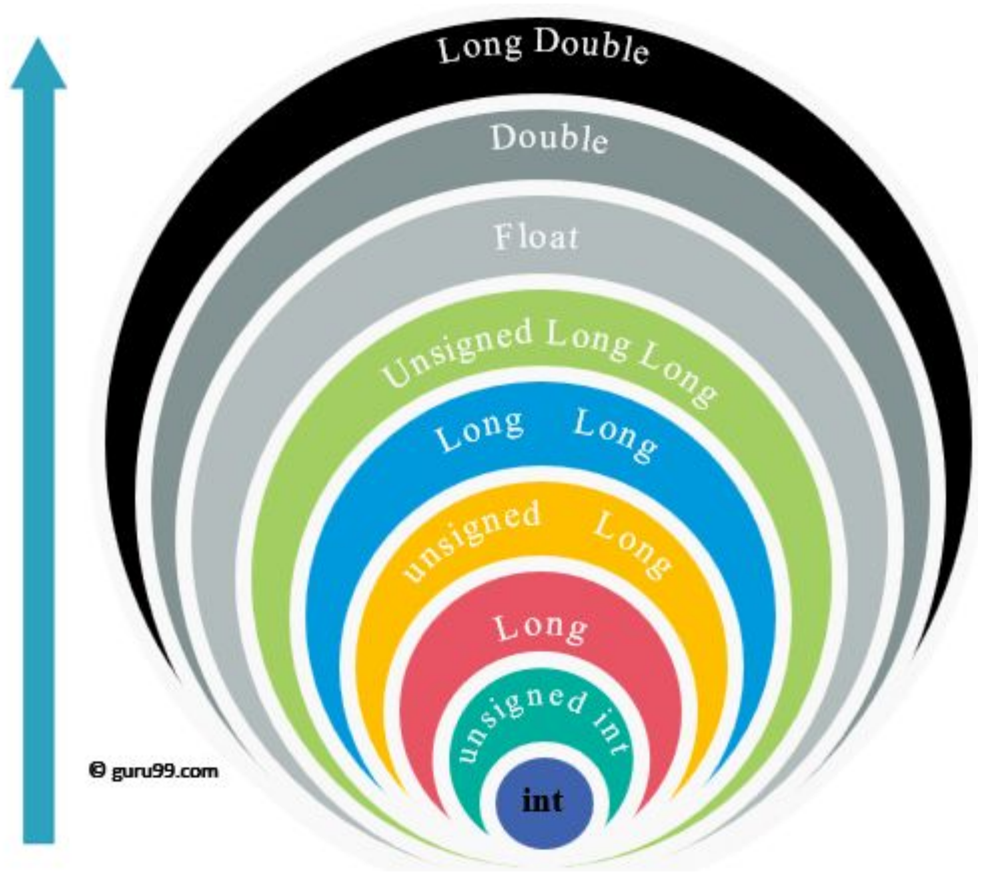
Explicit Conversion

```
int k = 5;

float y = ((float) k) / 10;
```

- k is explicitly converted to float.

Conversion Hierarchy



Questions

Which of the following is correct?

a. #include <stdio.h>
b. # include<stdio.h>
c. #include< stdio.h>
d. #include<stdio.h>

What is wrong with the following?

```
#include<stdio.h>
{
    int a = 10;
    printf(“%d”, a);
}
```