C Language Basics

ESCAPE SEQUENCES

- Character combination consisting of backslash(\) followed by letter or combination of digits.

Escape Sequence	Meaning
\a	Alert user by sounding
\b	Backspace
\n	New line
\t	Horizontal tab
\'	Prints single quote
\"	Prints double quote
\?	Prints question mark

- The backward slash is called escape character.

ASCII value

- A character variable holds ASCII value rather than character itself.
- ASCII values are between 0 to 127
- ASCII value of ‘A’ is 65 means , if you assign ‘A’ to a character variable , 65 is stored in that variable rather than character itself.
- Uppercase Alphabets – 65 to 90
- Lowercase Alphabets – 97 to 122

INPUT DATA

- Number of characters to be read can be restricted.
- Syntax : **%wd**
  - **w** is integer gives field width
  - **d** is data type character

  scanf ("%2d %4d", &num1, &num2);

const keyword

- Fixed value which cannot be changed in a program.
- It becomes read only variable.
- Can be any of the basic data types.
- Syntax: ***const*** datatype varname;

  datatype ***const*** varname;
- Declared with keyword ***const***
  - ***const float PI = 3.14;***
  - ***const int CODE1 = 101;*** (Integer constant)
  - ***int code2=102;*** (Integer variable)

volatile keyword

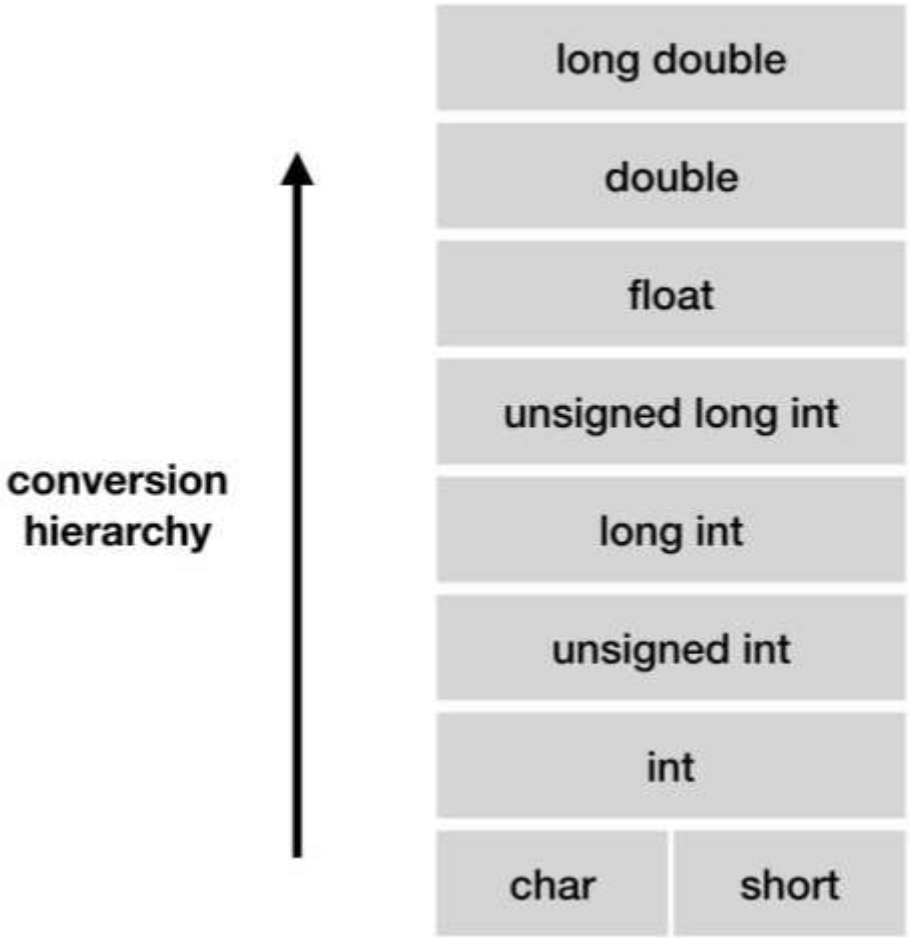
- If we declare the variable as ***volatile***, then it serves as a warning to compiler that it should not optimize the code containing this variable.
- Its value can be changed in ways that cannot be determined by the compiler.
- ***volatile*** variable should be always read from memory and its optimization is not possible.
- Example: **volatile int x;**

  y=x+x+x; //can't be optimize as 3*x

TYPE CASTING

- Converting the value of an expression of one data type into another data type.
- Two types
  - Implicit (by compiler)
  - Explicit (by programmer)

```
…
int x = 5, y = 2;
float a;
a = x / y;          //implicit
a = x/2.0;          //implicit
a = (float)x/y;     //explicit
…
```


conversion
hierarchy
long double
double
float
unsigned long int
long int
unsigned int
int
char
short

IMPLICIT TYPE CONVERSIONS

- If one operand is of LOWER RANK (LR) datatype & other is of HIGHER RANK (HR) datatype, then LOWER RANK will be converted to HIGHER RANK while evaluating the expression.

– int + int -> int

– float + float -> float

– int (promoted to float) + float -> float

OPERATION	RESULT
5 / 2	2
5.0 / 2	2.5
5 / 2.0	2.5
5.0 / 2.0	2.5
OPERATION	**RESULT**
2 / 5	0
2.0 / 5	0.4
2 / 5.0	0.4
2.0 / 5.0	0.4

TYPE CONVERSION IN ASSIGNMENTS

- **Type Promotion –**

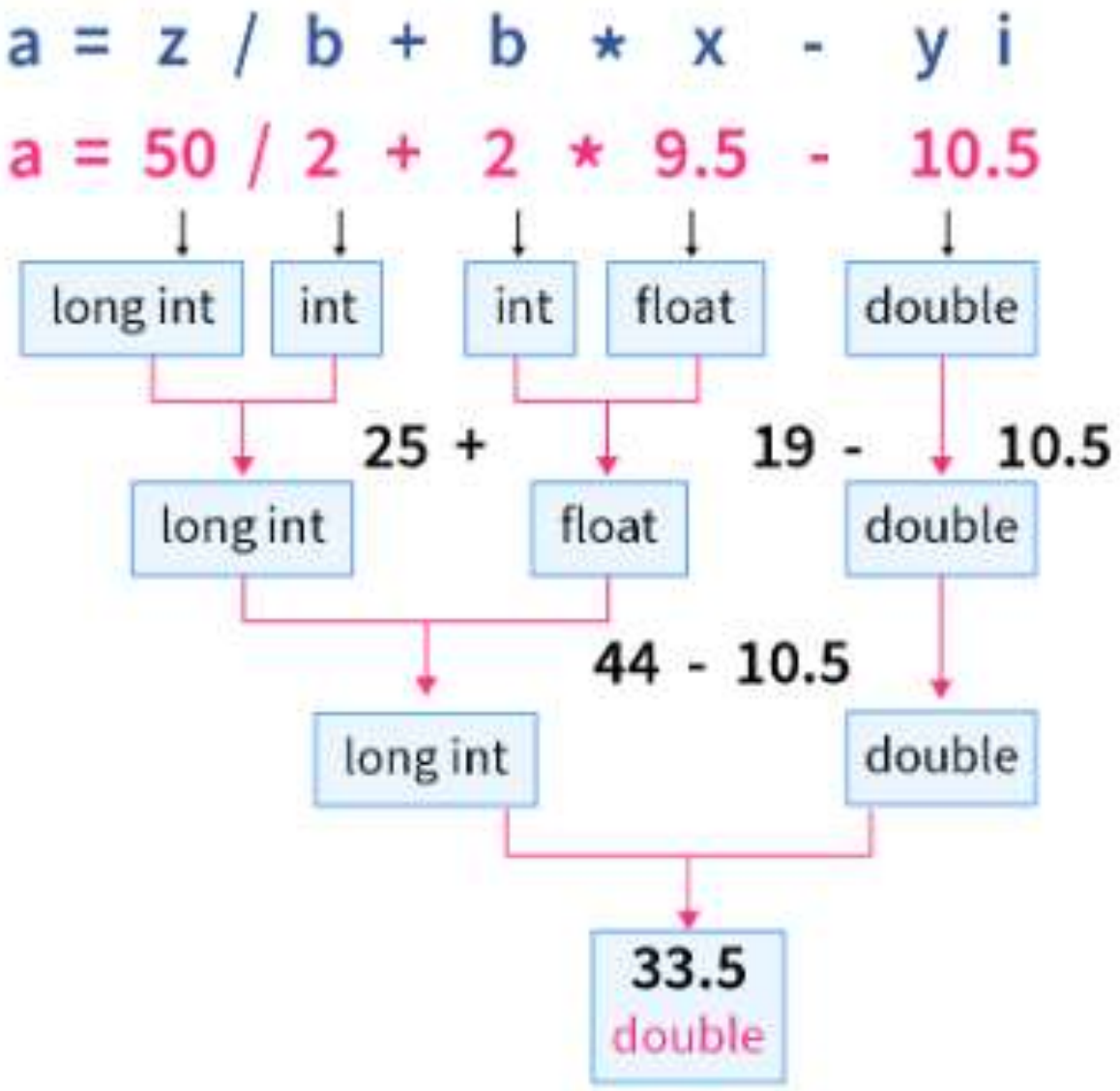
➢ LHS is HR and RHS is LR → int = char
LR is promoted → to HR while assigning

- **Type Demotion –**

➢ LHS is LR and RHS is HR → int = float
HR rank will be demoted → to LR. (Truncate)

int k ;
float a ;

ARITHMETIC INSTRUCTION	RESULT	ARITHMETIC INSTRUCTION	RESULT
k = 2 / 9	0	a = 2 / 9	0.0
k = 2.0 / 9	0	a = 2.0 / 9	0.222222
k = 2 / 9.0	0	a = 2 / 9.0	0.222222
k = 2.0 / 9.0	0	a = 2.0 / 9.0	0.222222
k = 9 / 2	4	a = 9 / 2	4.0
k = 9.0 / 2	4	a = 9.0 / 2	4.5
k = 9 / 2.0	4	a = 9 / 2.0	4.5
k = 9.0 / 2.0	4	a = 9.0 / 2.0	4.5

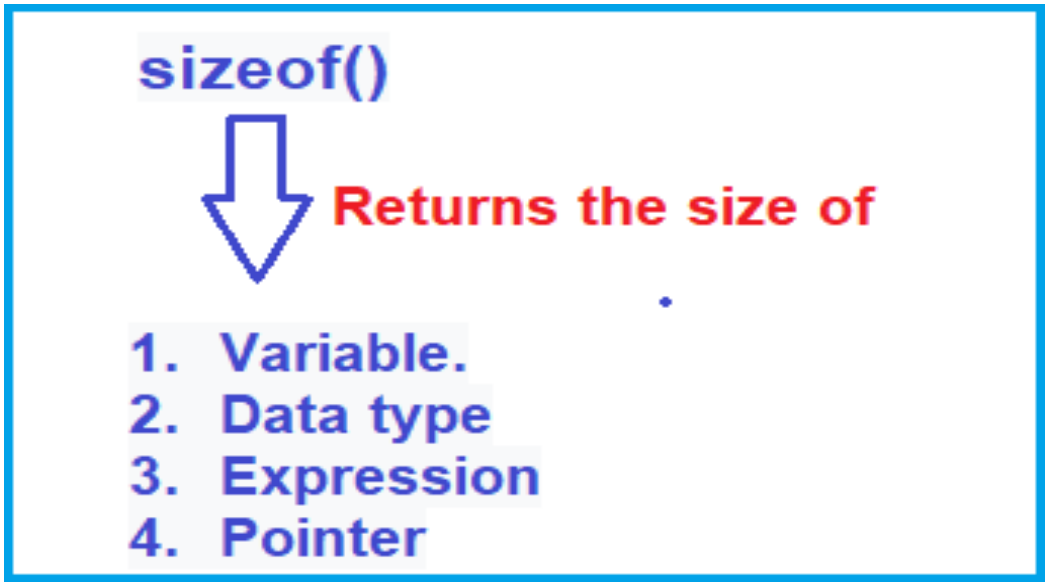

a = z / b + b * x - y i
a = 50 / 2 + 2 * 9.5 - 10.5
long int
int
int
float
double
25 +
19 -
10.5
long int
float
double
44 - 10.5
long int
double
33.5
double

sizeof() FUNCTION

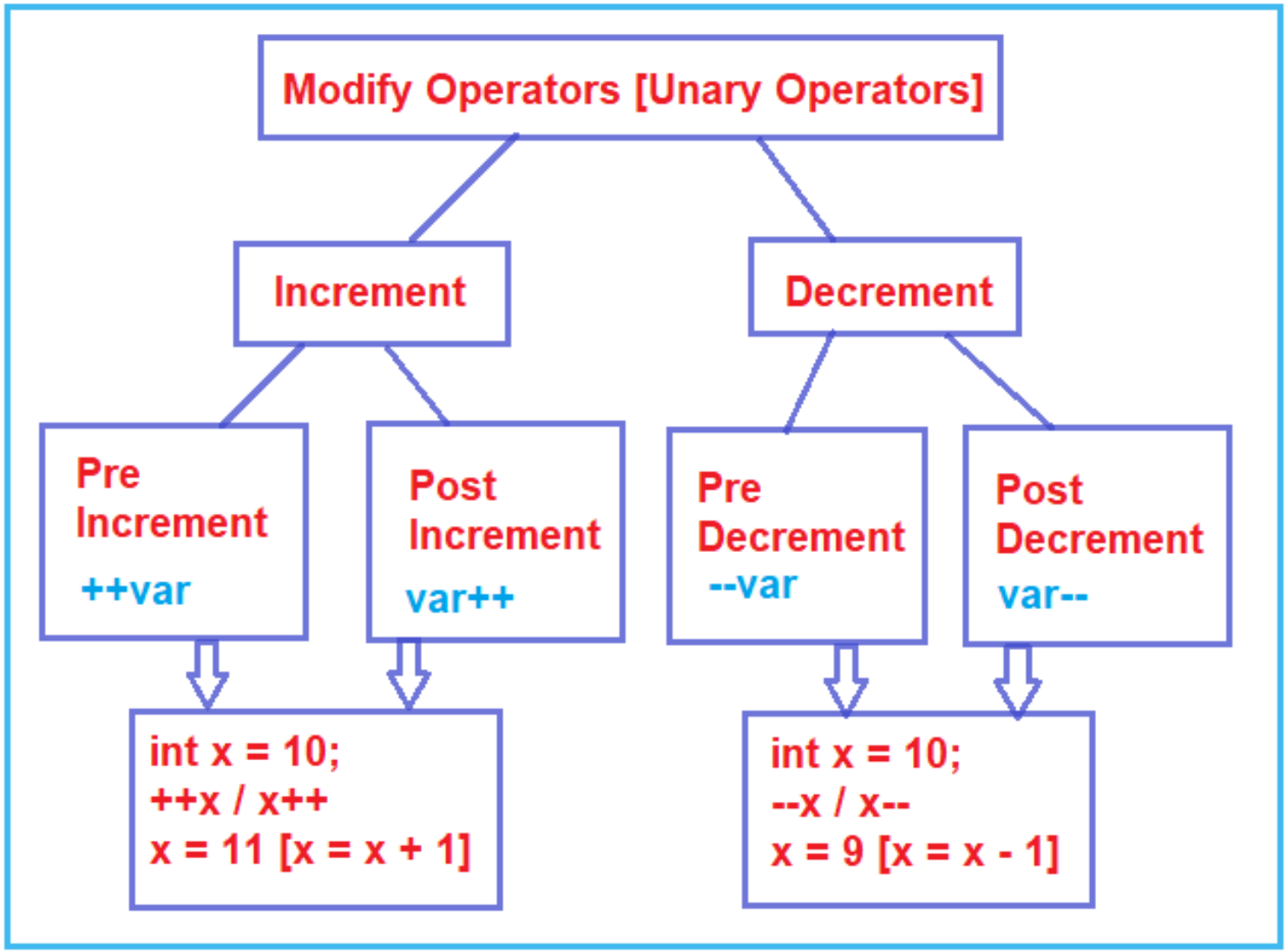
- It gives amount of storage in bytes

sizeof(expression);

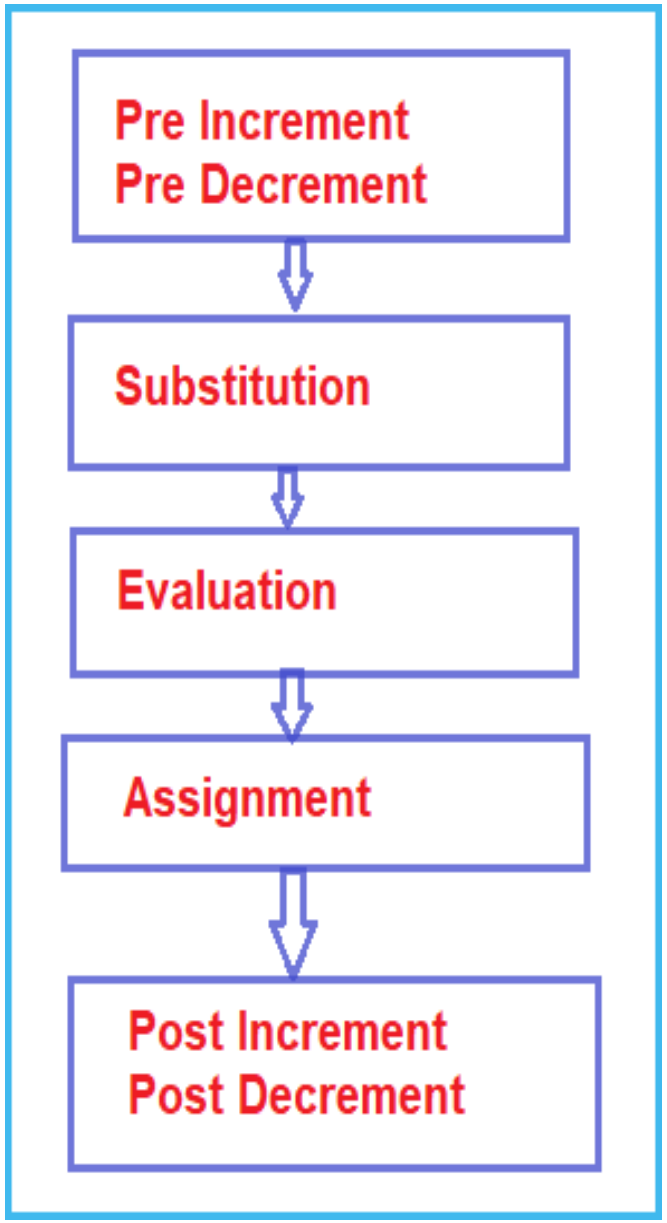
- Always returns integer value



INCREMENT and DECREMENT OPERATOR

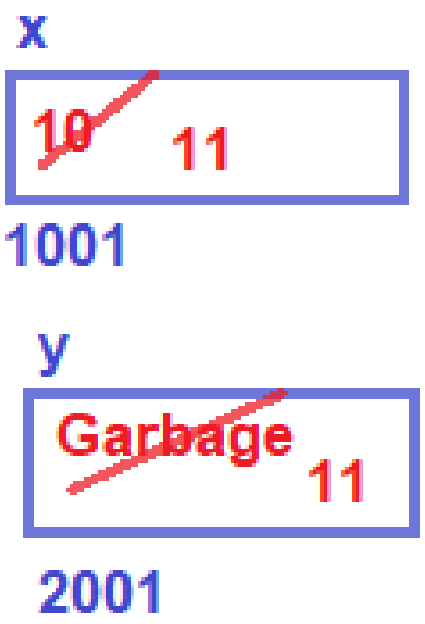


- If there is some pre-increment or pre decrement in the expression, that should execute first.
- The second step is to substitute the values in the expression.
- Once we substitute the values, in the third step we need to evaluate the expression.
- Followed by the Evaluation, an Assignment needs to be performed and the final step is post- increment or post decrement.

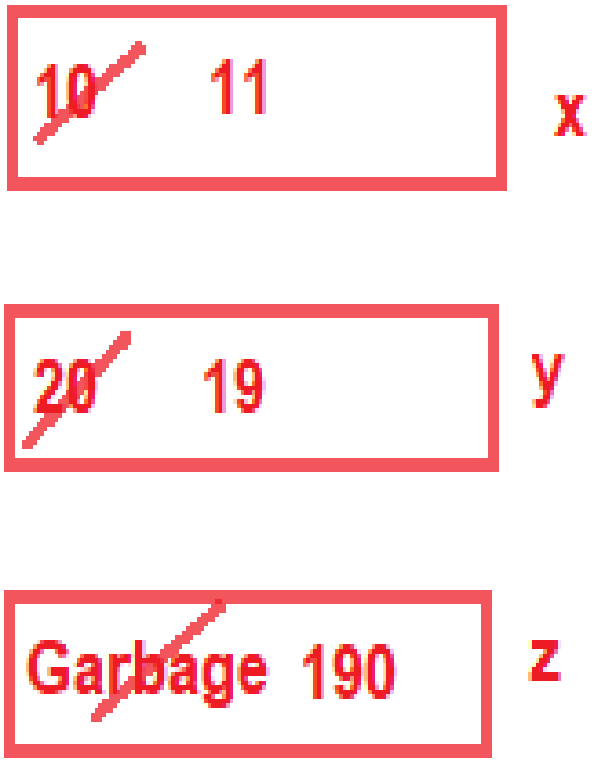


```
#include<stdio.h>
int main()
{
    int x = 10, y;
          11
    y = ++x;

    printf("%d %d", x, y);
    return 0;
}
```



```c
#include<stdio.h>
int main()
{
    int x = 10, y=20, z;
    z = x++ * --y;
    printf("%d %d %d", x, y, z);
    return 0;
}
```

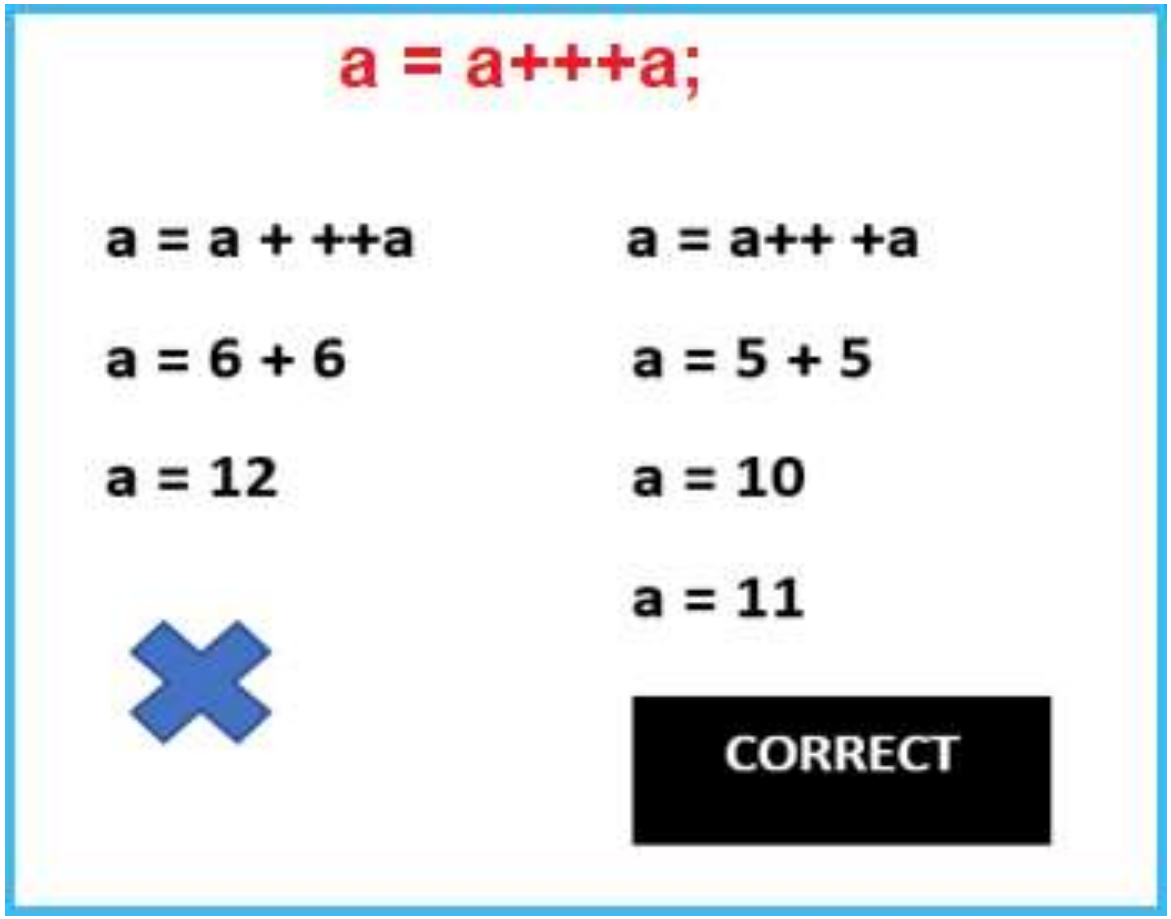


```c
#include<stdio.h>
int main()
{
    int x = 5;
    x = ++x + x++;
    printf("%d", x);
    return 0;
}
```

x

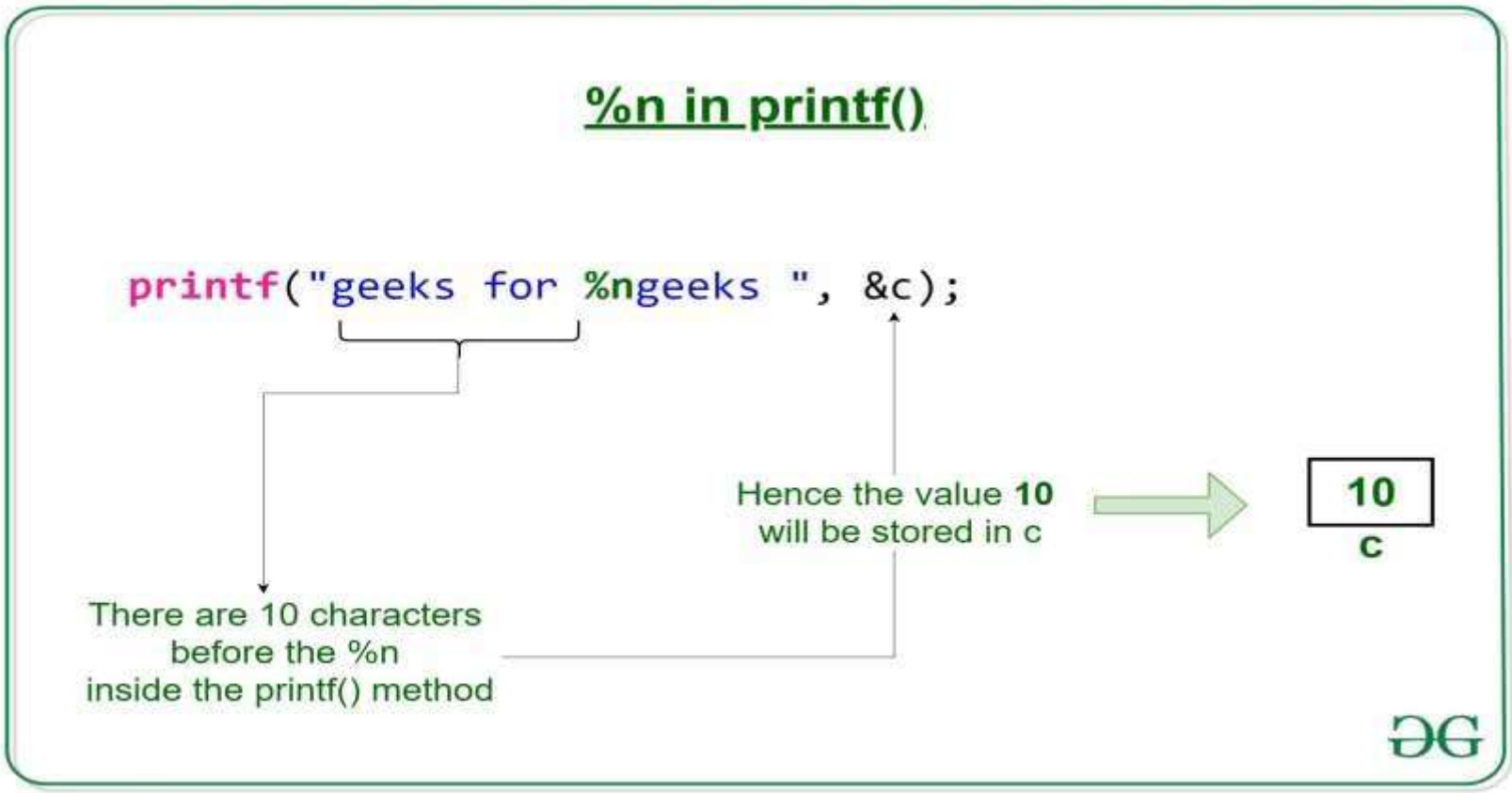
~~5~~ ~~6~~ ~~12~~ 13

-- The Unary operator has a priority greater than the arithmetic operator, so the compiler will execute the unary operator first.



%n Format Specifier

- In the case of printf() function the %n assign the number of characters printed by printf().



When we use the %n specifier in scanf() it will assign the number of characters read by the scanf() function until it occurs.