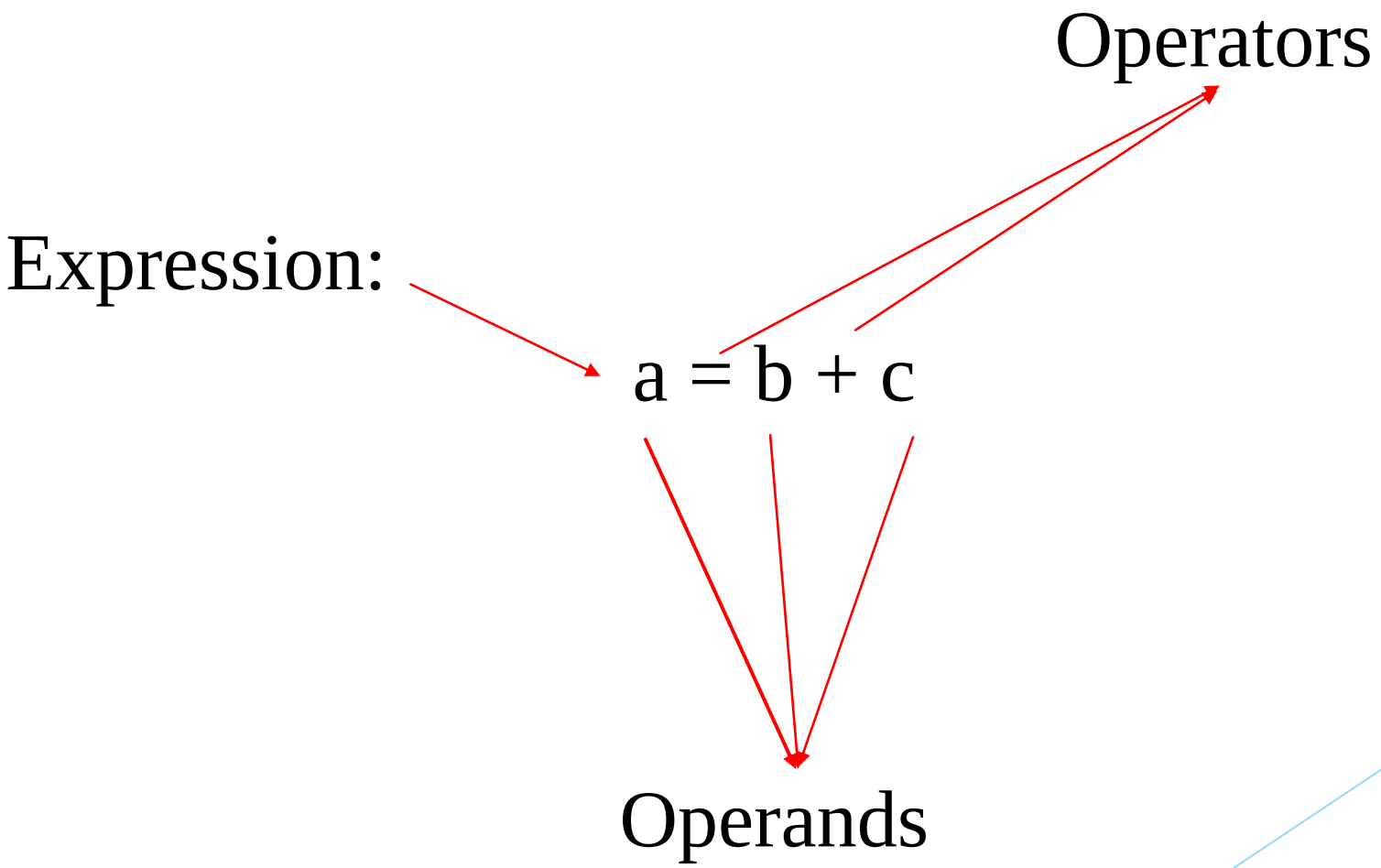
Chapter: 3.2
Operators and Expressions

Advanced College of Engineering And Management.

Topics to be covered

- Operators and Expressions
- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment/Decrement Operators
- Conditional Operators
- Bitwise Operators
- Comma Operators
- Sizeof() Operators
- Precedence and Associativity
- Type casting in C

Operators and Expression



- C supports a rich set of built in operators.
- An operator is a symbol that tells the computer to perform mathematical or logical manipulations
- Operators are broadly categorized into:
  - Arithmetic operators
  - Relational operators
  - Logical operators
  - Assignment operators
  - Increment and Decrement operators
  - Conditional operators
  - Bitwise operators
  - Special operators

Arithmetic operator

- performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).
- Arithmetic operators are:- +, -, *, /, %.

Example: value= a+b;

Relational Operators

- checks the relationship between two operands.
- If the relation is true, it returns 1; if the relation is false, it returns value 0.
- Relational operators are:- >, <, <=, >=, ==, !=

Example: a=10;

b=20;

b>a ⟶ returns 1

a>b ⟶ return 0

Logical Operator

- used in decision making in C programming.
- logical operator returns either 0 or 1 depending upon whether expression results true or false.
- Logical operators are: &&, ||, !.

Example: a=5;

b=10;

if(a==5 && b==10) ⟶ returns 1.

if(a==5 && b==20) ⟶ returns 0.

Assignment Operators

- used for assigning a value to a variable
- Assignment operators are: =, +=, -=, *=, /=, %=

Example: a+=b ⟶ a=a+b.

a*=b ⟶ a=a*b

Increment/Decrement operator

- Two operators: increment operator (++) and decrement operator (--).
- ++ operator increases the value of operand by 1.
- -- operator decreases the value of operand by 1.
- Increment operator are of two type: - pre increment operator (++a).
  -post increment operator (a++).
- Decrement operator are of two type: - pre decrement operator (--a).
  -post decrement operator (a--).

Example: (1) int a=10, b;

b = a++; ⟶ Assign value to b and increases value of a to 11.

i..e value of b will be 10, but value of a will be 11.

(2) int a=10, b;

b = ++a; ⟶ Increases value of a to 11 and assign value to b.

i..e value of a will be 11 and value of b will be 11.

Note: Decrement operators are same as increment operator but decrement operators decreases the value by 1.

Conditional Operator

- The operator (?:) is known as conditional operator.
- A conditional expression is written in the form:

  expression1? expression2: expression3;

  - expression1 is evaluated first.
  - If expression1 is true, the value of expression2 is the value of conditional expression.
  - If expression1 is false, the value of expression3 is the value of conditional expression.

- **Example:** (1) a = 10;

  b = 15;

  x = (a > b)? a: b;

  Here (a>b) is false so value of b will be assign to x, i..e value of x will be 15.

  (2) a = 10;

  b = 15;

  x = (a< b)? a: b;

  Here (a<b) is false so value of a will be assign to x, i..e value of x will be 10.

Bitwise Operators

- Performs operation in bit level i..e in binary form.
- Bitwise operators are:
  - Bitwise AND operator (&).
  - Bitwise OR operator (|).
  - Bitwise XOR operator (^).
  - Bitwise one's complement operator (~).
  - Bitwise Right shift operator (>>).
  - Bitwise Left shift operator (<<).

Bitwise AND operator (&)

- The output of bitwise AND is 1 if corresponding bits of two operand is 1, else 0.

Example: int a =12, b= 25, c;

c= a & b;

Here value of c will be 8.

a = 12 ↓(Binary) 0000 1100	b = 25 ↓(Binary) 0001 1001	c= a & b i..e a= 0000 1100 & b= 0001 1001 c= 0000 1000 So value of c is 8.

Bitwise OR Operator (|)

- The output of bitwise OR is 0 if corresponding bits of two operand is 0, else 1.

Example: int a =12, b= 25, c;

c= a | b;

Here value of c will be 29.

a = 12 ↓ (Binary) 0000 1100	b = 25 ↓ (Binary) 0001 1001	c= a | b i..e a= 0000 1100 | b= 0001 1001 c= 0001 1101 So value of c is 29.

Bitwise XOR Operator (^)

- The results of XOR operator is 1, if corresponding two bits are opposite, else 0.

Example: int a =12, b= 25, c;

c= a ^ b;

Here value of c will be 21.

a = 12 ↓ (Binary) 0000 1100	b = 25 ↓ (Binary) 0001 1001	c= a ^ b i..e a= 0000 1100 ^ b= 0001 1001 c= 0001 0101 So value of c is 21.

Bitwise 1's Complement Operator (~)

- It is unary operator and has effects of flipping bits i..e the results of ~ operator is 1, if given bit is 0 and 0 if given bits is 1.

Example: int a = 12, b;

b = ~a;

Here value of b will be 115.

a = 12 ↓ (Binary) 0000 1100	b= ~a i..e a= 0000 1100 so b= 1111 0011 So value of b is 115

Bitwise Right shift Operator (>>)

- Right shift operator shifts all bits towards right by certain number of specified bits.

Example: (1) int a= 212, b;

b = a >> 1

So value of b will be 106.

a= 212 ↓ (Binary) 1101 0100	Now a= 1 1 0 1 0 1 0 0 0 1 1 0 1 0 1 0 → Eleminated (Right shif by 1 i..e >>1 ↓ Added (Always added bit at MSB will be 0) So value of b will be 106

(2) int a= 212, b;

b = a >> 2

Here Right shift is performed 2 times.

i..e 1 1 0 1 0 1 0 0 → Given Question

0 1 1 0 1 0 1 0 (a >> 1) → Shifted Right by 1 bit.

b = 0 0 1 1 0 1 0 1 (a >> 2) → Again shifted Right by 1 bit.

i..e Value of b becomes 53.

Bitwise Left shift operator (<<)

- ✓ Left shift operator shifts all bits towards left by certain number of specified bits.

Example: (1) int a= 212, b;

b = a << 1

a= 212 ↓ (Binary) 1101 0100	Now a= 1 1 0 1 0 1 0 0 1 1 0 1 0 1 0 0 0 Always added bit at LSB will be zero

(2) int a= 212, b;

b = a << 2

Here shift is performed 2 times.

i..e 1 1 0 1 0 1 0 0 → Given Question

1 1 0 1 0 1 0 0 0 (a <<1) → Shifted Left by 1 bit.

b = 1 1 0 1 0 1 0 0 0 0(a <<2) → Again shifted Left by 1 bit.

i..e Value of b becomes 848.

Special Operators

- C supports some special operators such as comma operator, size of operator.
  - Comma Operator (,).
  - Sizeof Operator.

Comma Operator (,)

- The comma operator can be used to link the related expression together.
- Comma linked lists of expression are evaluated left to right and the value of right-most expression is the value of combined expression.

Example:

Value = (x = 10, y = 5, x + y),

Here, 10 is assigned to x and 5 is assigned to y and so expression x+y is evaluated as (10+5) i.e. 15.

Sizeof() Operator

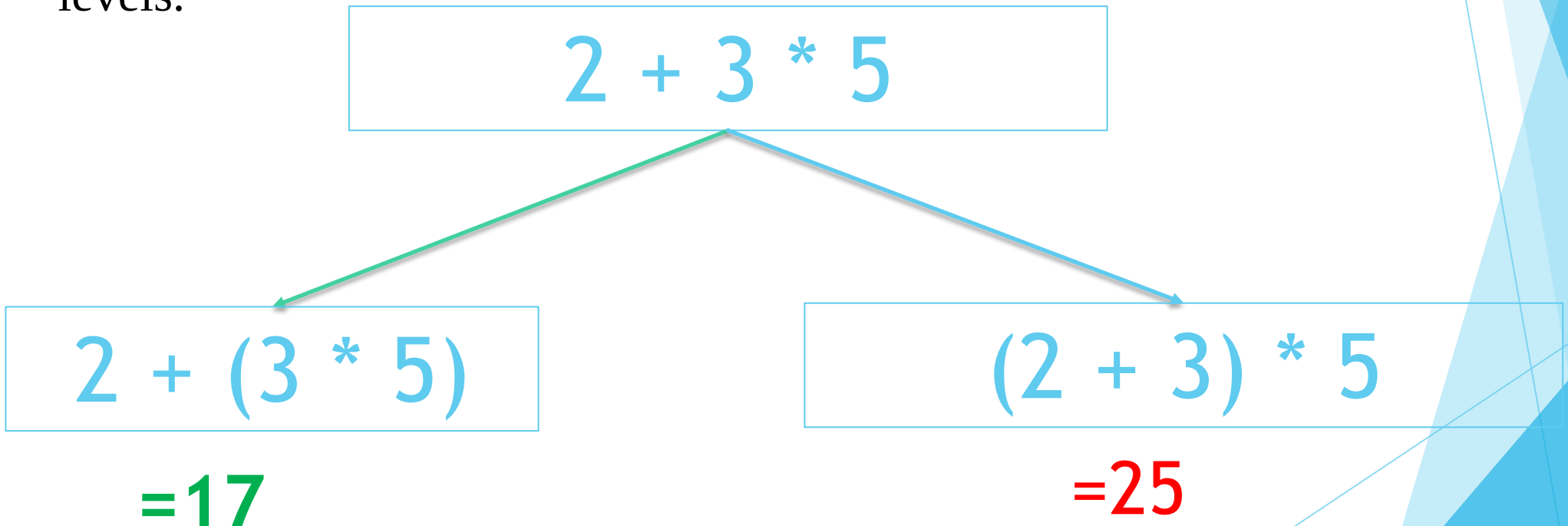
- The sizeof() operator is used with an operand to return the number of bytes it occupies.

Example:

(1) int a, value;
value = sizeof(a);
printf("value = %d", value);
↓
display value = 2
	(2) float a;
	int value;
	value = sizeof(a);
	printf("value = %d", value);
	↓
	display value = 4

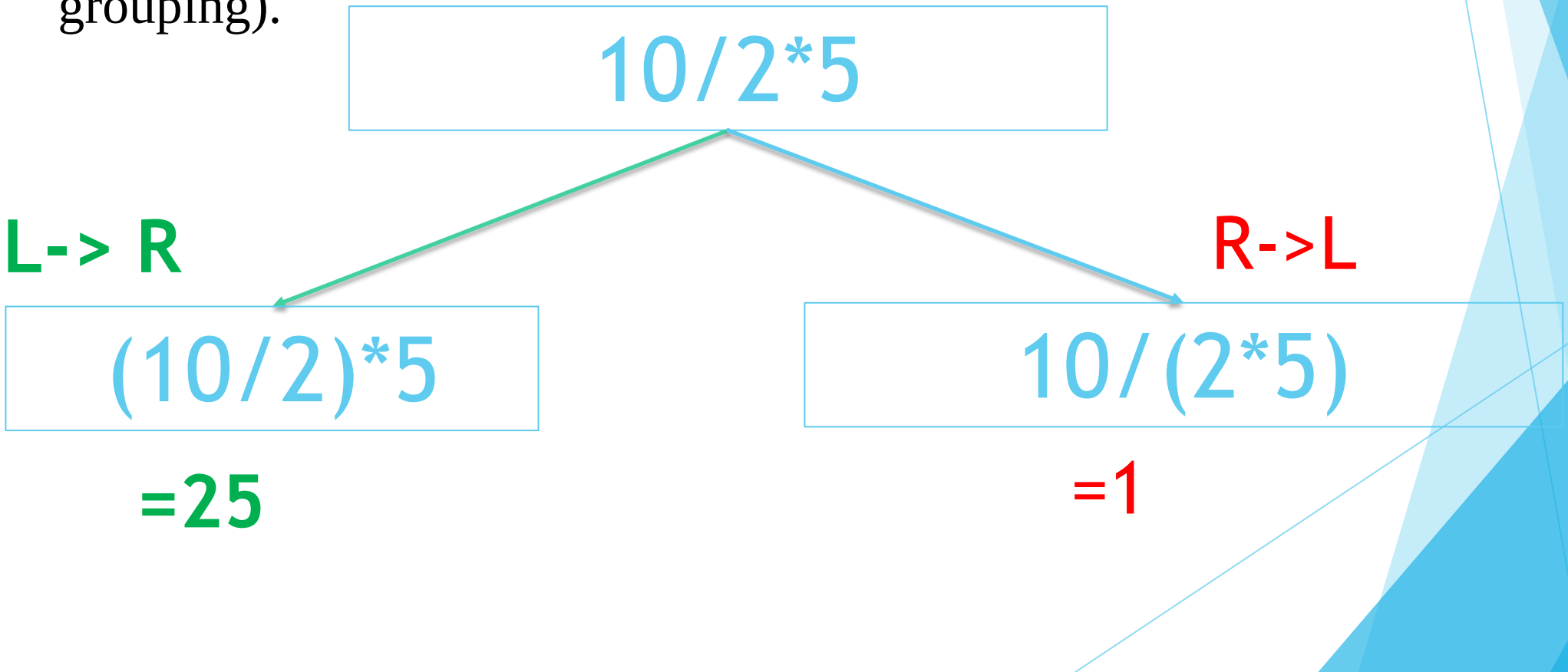
Precedence of Operators

- When expressions contain more than one operator, the order in which the operators are evaluated depends on their precedence levels.



Associativity of Operators

- If the precedence levels of operators are the same, then the order of evaluation depends on their associativity (or, grouping).



Operator precedence and Associativity

- A higher precedence operator is evaluated before a lower precedence operator.
- precedence and associativity of arithmetic operators:

Operators	**Decription**	**Associativity**	**Precedence Rank**
() []	Function call Array element Reference	Left -> Right	1
+ - ++ -- !	Unary Plus Unary minus Increment Decrement Logical negation	Right -> Left	2

~ * & sizeof	Ones complement Pointer reference Address Sizeof operator	Right -> Left	2
* / %	Multiplication Division Modulus	Left -> Right	3
+ -	Addition Subtraction	Left -> Right	4
<< <<	Left shift Right Shift	Left -> Right	5
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left -> Right	6

== !=	Equality Inequality	Left -> Right	7
&	Bitwise AND		8
^	Bitwise XOR		9
|	Bitwise OR		10
&&	Logical AND		11
||	Logical OR		12
?:	Conditional operator		13
= += -= *= /= %=	Assignment operator	Right -> Left	14
,	Comma operator	Left -> Right	15

Type casting in C

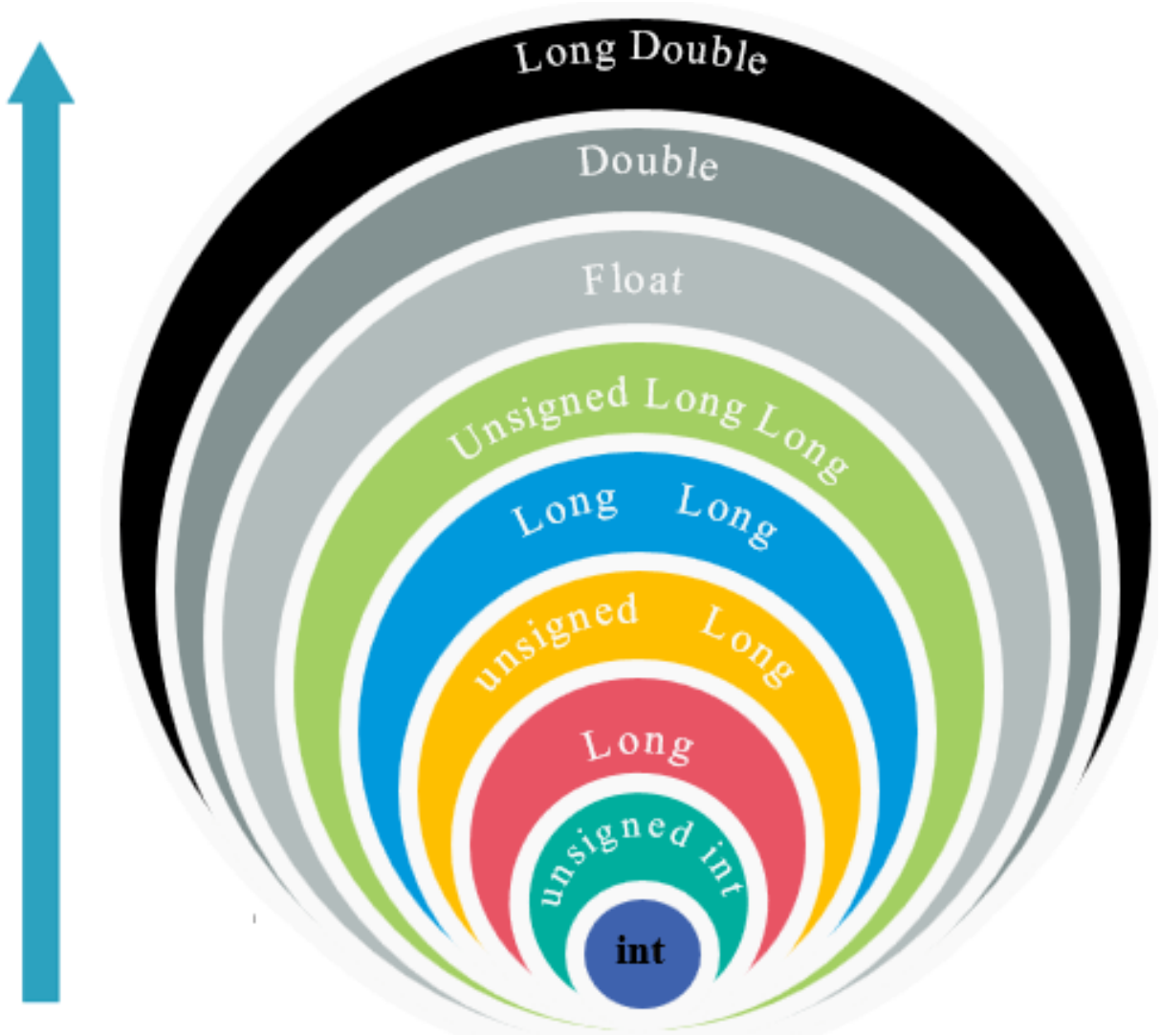
- The process of converting one predefined type into another is called type conversion.
- Two types: Implicit type conversion
  Explicit type conversion

Implicit Type Conversion

- type conversion is performed automatically by the compiler.
- Lower data type are converted to higher data type automatically.

Example:

```
#include<stdio.h>
int main()
{   int x = 10;    // integer x
    char y = 'a';  // character c  and its ascii value is 97
    x = x + y;     ⟶ char type is implicitly converted to int type
    printf(" Value of x=%d", x);   ⟶ Gives result value =107
}
```


Long Double
Double
Float
Unsigned Long Long
Long Long
unsigned Long
Long
unsigned int
int

Explicit type conversion

- The type conversion performed by the programmer by posing the data type of the expression of specific type

Example:

```
#include<stdio.h>
int main()
{
    float x = 1.2;
    int sum;
    sum = (int)x + 1;
    printf("sum = %d", sum);
    return 0;
}
```

Explicit conversion from float to int

Gives result sum= 2

Practice Questions

- WAP To Convert Celsius to Fahrenheit [C=F-32/1.8]
- WAP To convert seconds as input to minutes
- WAP to take marks of 5 subjects from student and calculate Total marks and percentage.
- WAP to enter 4- digit number and find the sum of first and last digit of the number.