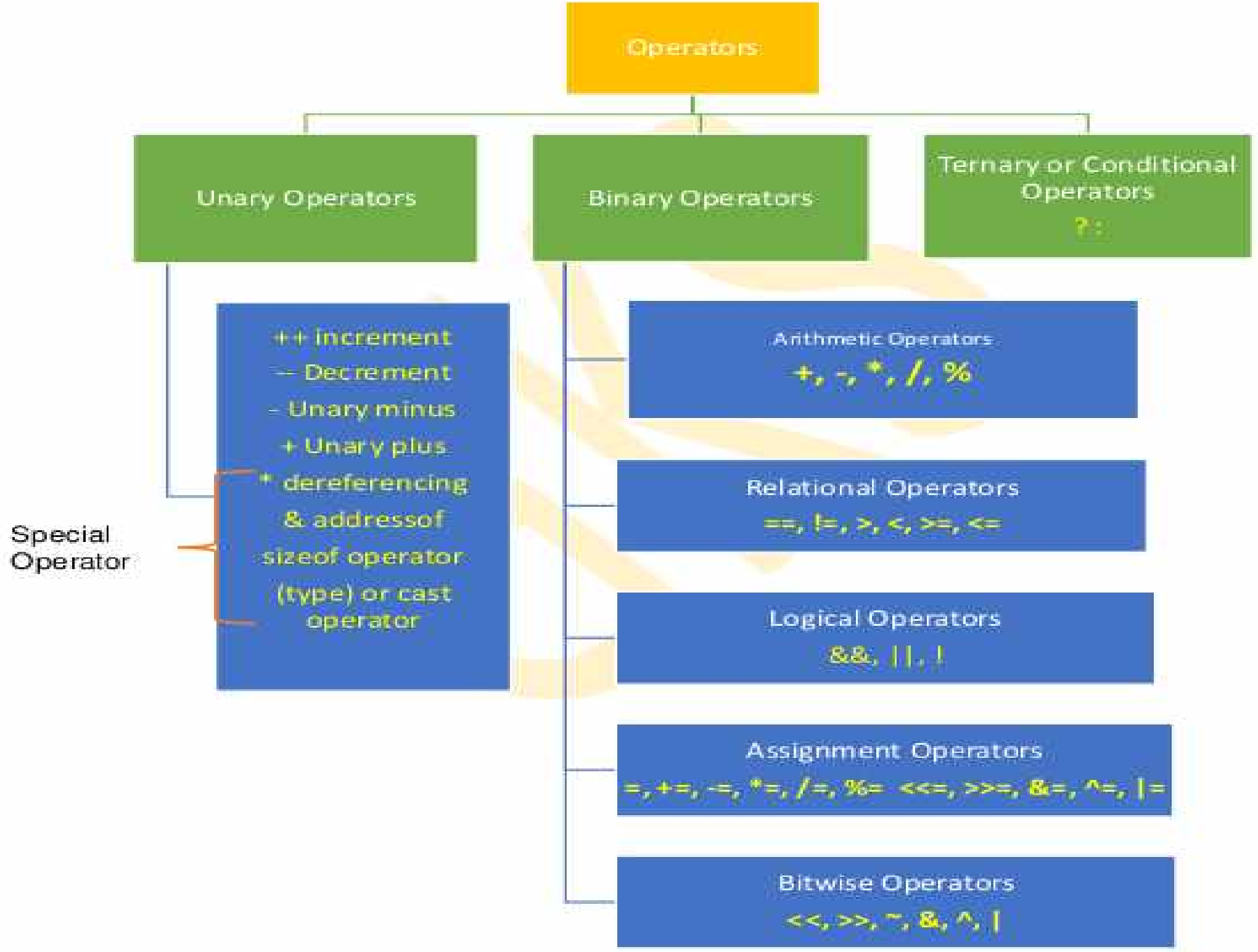
Operators in C

Operators in C

- An operator is a symbol that tells the compiler to perform a certain mathematical or logical manipulation.
- Operators are used in programs to manipulate data and variables.
- C operators can be classified into the following types:
- Arithmetic operators
- Relational operators
- Logical operators
- Bitwise operators
- Assignment operators
- Conditional operators
- Special operators


Operators
Unary Operators
Binary Operators
Ternary or Conditional Operators
? :
++ increment
-- Decrement
- Unary minus
+ Unary plus
* dereferencing
& addressof
sizeof operator
(type) or cast operator
Special Operator
Arithmetic Operators
+, -, *, /, %
Relational Operators
==, !=, >, <, >=, <=
Logical Operators
&&, ||, !
Assignment Operators
=, +=, -=, *=, /=, %= <<=, >>=, &=, ^=, |=
Bitwise Operators
<<, >>, ~, &, ^, |

Arithmetic Operators

Operator	Description	Example
+	adds two operands (values)	a+b
-	subtract second operands from first	a-b
*	multiply two operands	a*b
/	divide numerator by the denominator	a/b
%	**modulus operator**, it returns the remainder of the division of two operands as the result	a%b
++	**Increment operator** - increases integer value by one. This operator needs only a single operand.	a++ or ++a
--	**Decrement operator** - decreases integer value by one. This operator needs only a single operand.	--b or b--

Arithmetic Operators Example

```
#include <stdio.h>
int main() {
    int a = 50, b = 23, result;
    // addition
    result = a+b;
    printf("Addition of a & b = %d \n",result);
    // subtraction
    result = a-b;
    printf("Subtraction of a & b = %d \n",result);
    // multiplication
    result = a*b;
    printf("Multiplication of a & b = %d \n",result);
    // division
    result = a/b;
    printf("Division of a & b = %d \n",result);
    return 0;
}
```

Modulus Operator

```
#include <stdio.h>

int main() {

   int a = 23, b = 20, result;

   // Using Modulus operator
   result = a%b;

   printf("result = %d",result);

   return 0;

}
```

Examples

Expression	%d	%f
5/2	2	2.0000
2/5	0	0.0000
5.0/2.0	2	2.50000
2.0/5.0	0	0.40000
5.0/2	2	2.5000

Relational Operators

• The relational operators (or comparison operators) are used to check the relationship between two operands. It checks whether two operands are equal or not equal or less than or greater than, etc.
• It returns 1 if the relationship checks pass, otherwise, it returns 0.
• Relational operators are used in decision making and loops.

Operator	Description	Example (a and b, where a = 10 and b = 11)
==	Check if two operands are equal	a == b, returns 0
!=	Check if two operands are not equal.	a != b, returns 1 because a is not equal to b
>	Check if the operand on the left is greater than the operand on the right	a > b, returns 0
<	Check operand on the left is smaller than the right operand	a < b, returns 1
>=	check left operand is greater than or equal to the right operand	a >= b, returns 0
<=	Check if the operand on left is smaller than or equal to the right operand	a <= b, returns 1

Logical Operators

These operators are used to perform logical operations and used with conditional statements like if-else statements.

- With AND operator, only if both operands are true, the result is true.
- With the OR operator, if a single operand is true, then the result will be true.
    - The NOT operator changes true to false, and false to true.

Operator	Description	Example (a and b, where a = 1 and b = 0)
&&	Logical AND	a && b, returns 0
||	Logical OR	a || b, returns 1
!	Logical NOT	!a, returns 0

Bitwise Operators

- Bitwise operators perform manipulations of data at the bit level.
- These operators also perform the shifting of bits from right to left.
- Bitwise operators are not applied to float or double, long double, void, etc.
- & Bitwise AND | Bitwise OR ^ Bitwise Exclusive OR (XOR)
- ~ One's complement (NOT) >> Shift right << Shift left

- a = 00010000
- b = 2
- a << b = 01000000
- a >> b = 00000100

Assignment Operators

- The sssignment operators are used to assign value to a variable.
- When we combine the arithmetic operators with the assignment operator =, then we get the shorthand form of all the arthmetic operators.

Operator	Description	Example (a and b are two variables, with where a=10 and b=5)
=	assigns values from right side operand to left side operand	a=b, a gets value 5
+=	adds right operand to the left operand and assign the result to left operand	a+=b, is same as a=a+b, value of a becomes 15
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b, is same as a=a-b, value of a becomes 5
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b, is same as a=a*b, value of a becomes 50
/=	divides left operand with the right operand and assign the result to left operand	a/=b, is same as a=a/b, value of a becomes 2
%=	calculate modulus using two operands and assign the result to left operand	a%=b, is same as a=a%b, value of a becomes 0

Ternary Operator (?)

The ternary operator, also known as the conditional operators in the C language can be used for statements of the form if-then-else.

The basic syntax for using ternary operator is:

```
(Expression1) ? Expression2 : Expression3 ;
```

Here is how it works:

- The question mark ? in the syntax represents the if part.
- The first expression (expression 1) returns either true or false, based on which it is decided whether (expression 2) will be executed or (expression 3)
- If (expression 1) returns true then the (expression 2) is executed.
- If (expression 1) returns false then the expression on the right side of : i.e (expression 3) is executed.

Ternary Operator Example -

```
// 1. Find number is positive or negative

#include<stdio.h>

void main()

{

int num;

printf("Enter a number: ");

scanf("%d", &num);

(num>=0)?printf("Positive."):printf("Negative");

}
```

```
// 2.

#include <stdio.h>

int main() {

  int a = 20, b = 20, result;

  /* Using ternary operator

     - If a == b then store a+b in result

     - otherwise store a-b in result

  */

  result = (a==b)?(a+b):(a-b);

  printf("result = %d",result);

  Return 0;

}
```

Special Operators

Operator	Description	Example
`sizeof`	returns the **size(length in bytes) of entity**, for eg. a variable or an array, etc.	`sizeof(x)` will return size of the variable `x`
`&`	returns the memory address of the variable	`&x` will return address of the variable `x`
`*`	represents pointer to an object. The `*` operator returns the value stored at a memory address.	`m = &x` (memory address of variable `x`) `*m` will return the value stored at memory address `m`
`.` (dot) operator	used to access individual elements of a C structure or C union.	If `emp` is a structure with an element `int age` in it, then `emp.age` will return the value of age.
`->` (arrow) operator	used to access structure or union elements using a pointer to structure or union.	If `p` is a pointer to the `emp` structure, then we can access `age` element using `p->age`
`[]` operator	used to access array elements using indexing	if `arr` is an array, then we can access its values using `arr[index]`, where `index` represents the array index starting from zero

Example -

```
#include <stdio.h>

int main() {

    int a = 20;
    char b = 'B';
    float c = 494.224;

    // sizeof operator
    printf("Size of a is: %d \n", sizeof(a));
    printf("Size of b is: %d \n", sizeof(b));
    printf("Size of c is: %d \n", sizeof(c));

    // & operator
    printf("Memory address of a: %d \n", &a);

    return 0;

}
```

Summary of Operators

Category	Operators	Associativity
Highest	( ) fun. call [ ] Subscript → pointer to member • Member access.	Left to Right.
unary	! Not ~ 1's complement − unary minus + unary plus ++/-- pre/post incr & address of * value at addrs sizeof typecast	Right to Left
Arithmetic	* mul / div % (Remainder) modulus	Left to Right
	+ plus − minus.	L to R
Bitwise	<< (left shift)	L to R
	>> (Right shift)	L to R
Relational	< less than <= less than or equal to > greater than >= greater than equal to	L to R
	== Equality != Not Equal to	L to R
Bitwise	& Bitwise and	L to R
	^ Bitwise Exclusive OR	L to R
	| Bitwise OR	L to R

Summary of Operators (Cont..)

Logical	&& (Logical And)	L to R
	|| (Logical OR)	L to R
Conditional & Ternary operator	? :	Right to Left
Assignment	= Equal to, *= /= %= += -= <<= >>= &= ^= |= Assignment	R to L
Comma	,	L to R
	++ / -- post increment / decrement.	Right to Left.