



Operators and Expressions

Session 3



Objectives

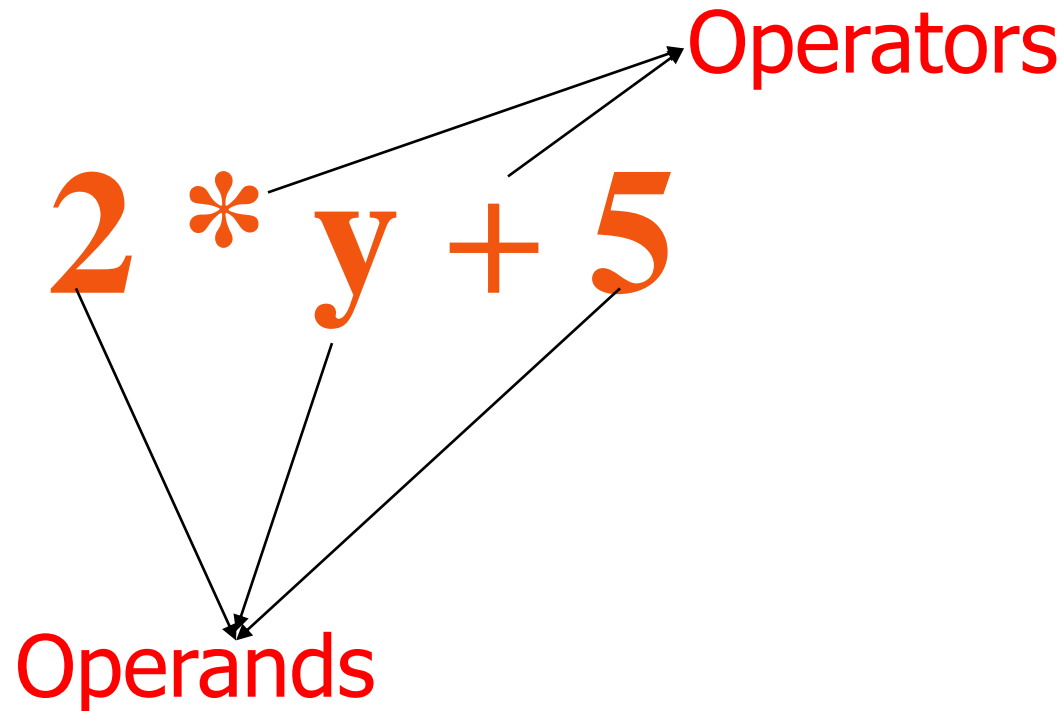
- Explain Assignment Operator
- Understand Arithmetic Expressions
- Explain Relational and Logical Operators
- Understand Bitwise logical operators and expressions
- Explain casts
- Understand Precedence of Operators



Expressions

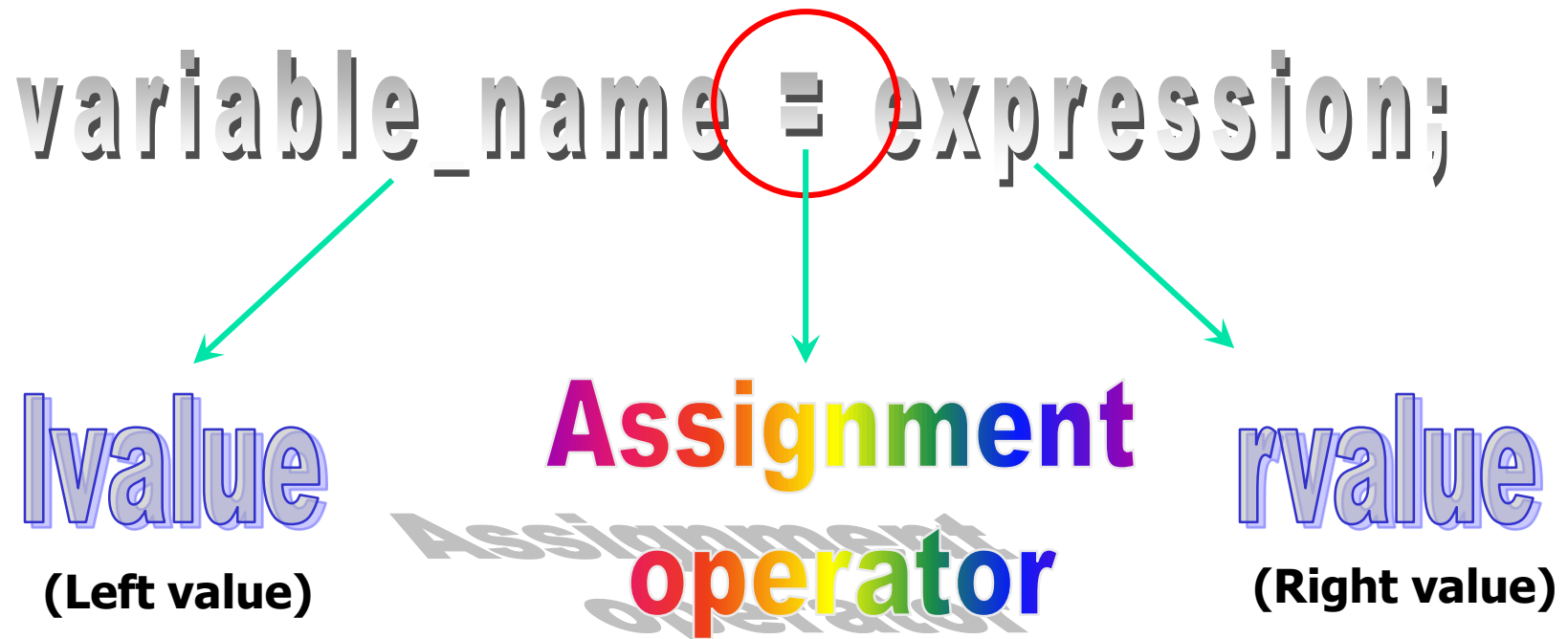
Combination of Operators and Operands

Example



The Assignment Operator

The assignment operator(=) can be used with any valid C expression





Multiple Assignment

Many variables can be assigned the same value in a single statement

```
a = b = c = 10;
```



However, you cannot do this :

```
int a = int b = int b = int c = 10
```





Operators

4 Types



Arithmetic



Logical



Relational



Bitwise



Arithmetic Operators

Arithmetic operators are used to perform numerical operations.

They are divided into two classes: **unary** and **binary** arithmetic operators

Unary Operators	Action	Binary Operators	Action
-	Unary minus	+	Addition
++	Increment	-	Subtraction
--	Decrement	*	Multiplication
		%	Modulus
		/	Division
		^	Exponentiation



Arithmetic Expressions

Mathematical expressions can be expressed in C using arithmetic operators

Examples

`++i % 7`

`5 + (c = 3 + 8)`

`a * (b + c/d)22`

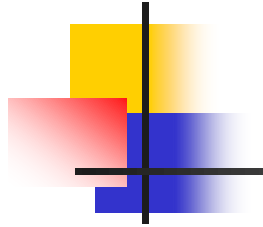


Relational Operators

Relational operators are used to test the relationship between two variables, or between a variable and a constant

Operator	Relational Operators Action
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Equal
!=	Not equal

Example: `if (year % 4 == 0)`



Logical Operators

Logical operators are symbols that are used to combine or negate expressions containing relational operators

Operator	Logical Operators Action
&&	AND
	OR
!	NOT

Example: if (a>10) && (a<20)

Expressions that use logical operators return zero for false, and 1 for true



Bitwise Logical Operators-1

Processes data after converting number to its binary equivalent. (Bit wise representation)

AND (NUM1 & NUM2)	Return 1 if both the operands are 1
OR (NUM1 NUM2)	Returns 1 if bits of either of the operand are 1
NOT (~ NUM1)	Reverses the bits of its operand (from 0 to 1 and 1 to 0)
XOR (NUM1 ^ NUM2)	Returns 1 if either of the bits in an operand is 1 but not both



Bitwise Logical Operators-2

Example

$10 \& 15 \rightarrow 1010 \& 1111 \rightarrow 1010 \rightarrow 10$

$10 | 15 \rightarrow 1010 | 1111 \rightarrow 1111 \rightarrow 15$

$10 \wedge 15 \rightarrow 1010 \wedge 1111 \rightarrow 0101 \rightarrow 5$

$\sim 10 \rightarrow \sim 1010 \rightarrow 1011 \rightarrow -11$



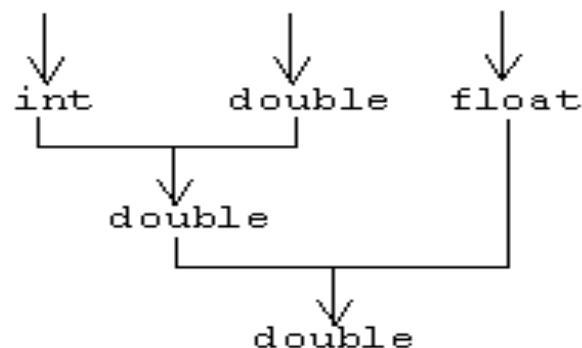
Type Conversion

The **automatic type conversions** for evaluating an expression are tabulated below.

- char** and **short** are converted to **int** and **float** is converted to **double**.
- If either operand is **double**, the other is converted to **double**, and the result is **double**.
- If either operand is **long**, the other is converted to **long** the result is **double**.
- If either operand is **unsigned**, the other is also converted to **unsigned** and the result is also **unsigned**.
- Otherwise all that are left are the operands of type **int**, and the result is **int**.

Example

```
char ch;  
int i;  
float f;  
double d;  
result = (ch/i) + (f*d) - (f+i);
```





Casts

An expression can be forced to be of a certain type by using a cast. The general syntax of cast:

(type) cast

type → any valid C data type

Example:

```
float x,f;
```

```
f = 3.14159;
```

```
x = (int) f; , the value of x will be 3 (integer)
```

The integer value returned by **(int)f** is converted back to floating point when it crossed the assignment operator. The value of f itself is not changed.



Precedence Of Operators-1

- Precedence establishes the hierarchy of one set of operators over another when an arithmetic expression is to be evaluated
- It refers to the order in which C evaluates operators
- The precedence of the operators can be altered by enclosing the expressions in parentheses

Operator Class	Operators	Associativity
Unary	- ++ --	Right to Left
Binary	^	Left to Right
Binary	* / %	Left to Right
Binary	+ -	Left to Right
Binary	=	Right to Left



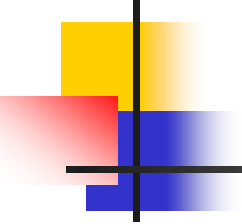
Precedence Of Operators-2

Example :

$-8 * 4 \% 2 - 3$

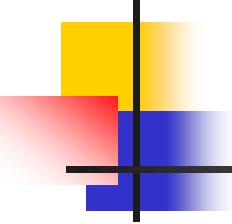
Sequence	Operation done	Result
1.	- 8 (unary minus)	negative of 8
2.	- 8 * 4	- 32
3.	- 32 % 2	16
4.	16-3	13

Precedence between comparison Operators



Always evaluated from left to right

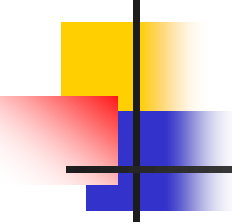




Precedence for Logical Operators-1

Precedence	Operator
1	NOT
2	AND
3	OR

When multiple instances of a logical operator are used in a condition, they are evaluated from right to left



Precedence for Logical Operators-2

Consider the following expression
False OR True AND NOT False AND True

This condition gets evaluated as shown below:

False OR True AND [NOT False] AND True

NOT has the highest precedence.

False OR True AND [True AND True]

AND is the operator of the highest precedence and operators of the same precedence are evaluated from right to left

False OR [True AND True]

[False OR True]

True

Precedence among Operators-1

When an equation uses more than one type of operator then the order of precedence has to be established with the different types of operators

Precedence	Type of Operator
1	Arithmetic
2	Comparison
3	Logical

Precedence among Operators-2



Consider the following example:

$2*3+4/2 > 3 \text{ AND } 3<5 \text{ OR } 10<9$

The evaluation is as shown:

$[2*3+4/2] > 3 \text{ AND } 3<5 \text{ OR } 10<9$

First the arithmetic operators are dealt with

$[[2*3]+[4/2]] > 3 \text{ AND } 3<5 \text{ OR } 10<9$

$[6+2] > 3 \text{ AND } 3<5 \text{ OR } 10<9$

$[8 > 3] \text{ AND } [3 < 5] \text{ OR } [10 < 9]$



Precedence among Operators-3

Next to be evaluated are the comparison operators all of which have the same precedence and so are evaluated from left to right

True AND True OR False

The last to be evaluated are the logical operators.
AND takes precedence over OR

[True AND True] OR False

True OR False



Changing Precedence-1

- Parenthesis () has the highest level of precedence
- The precedence of operators can be modified using parenthesis ()
- Operator of lower precedence with parenthesis assume highest precedence and gets executed first
- In case of nested Parenthesis ((())) the inner most parenthesis gets evaluated first
- An expression consisting of many set of parenthesis gets processed from left to right



Changing Precedence-2

Consider the following example:

$5+9*3^2-4 > 10 \text{ AND } (2+2^4-8/4 > 6 \text{ OR } (2<6 \text{ AND } 10>11))$

The solution is:

1. $5+9*3^2-4 > 10 \text{ AND } (2+2^4-8/4 > 6 \text{ OR } (\text{True AND False}))$

The inner parenthesis takes precedence over all other operators and the evaluation within this is as per the regular conventions

2. $5+9*3^2-4 > 10 \text{ AND } (2+2^4-8/4 > 6 \text{ OR False})$



Changing Precedence-3

3. $5+9*3^2-4 > 10$ AND $(2+16-8/4 > 6$ OR False)

Next the outer parentheses is evaluated

4. $5+9*3^2-4 > 10$ AND $(2+16-2 > 6$ OR False)

5. $5+9*3^2-4 > 10$ AND $(18-2 > 6$ OR False)

6. $5+9*3^2-4 > 10$ AND $(16 > 6$ OR False)

7. $5+9*3^2-4 > 10$ AND (True OR False)

8. $5+9*3^2-4 > 10$ AND True



Changing Precedence-4

9. $5+9*9-4>10$ AND True

The expression to the left is evaluated as per the conventions

10. $5+81-4>10$ AND True

11. $86-4>10$ AND True

12. $82>10$ AND True

13. True AND True

14. **True**