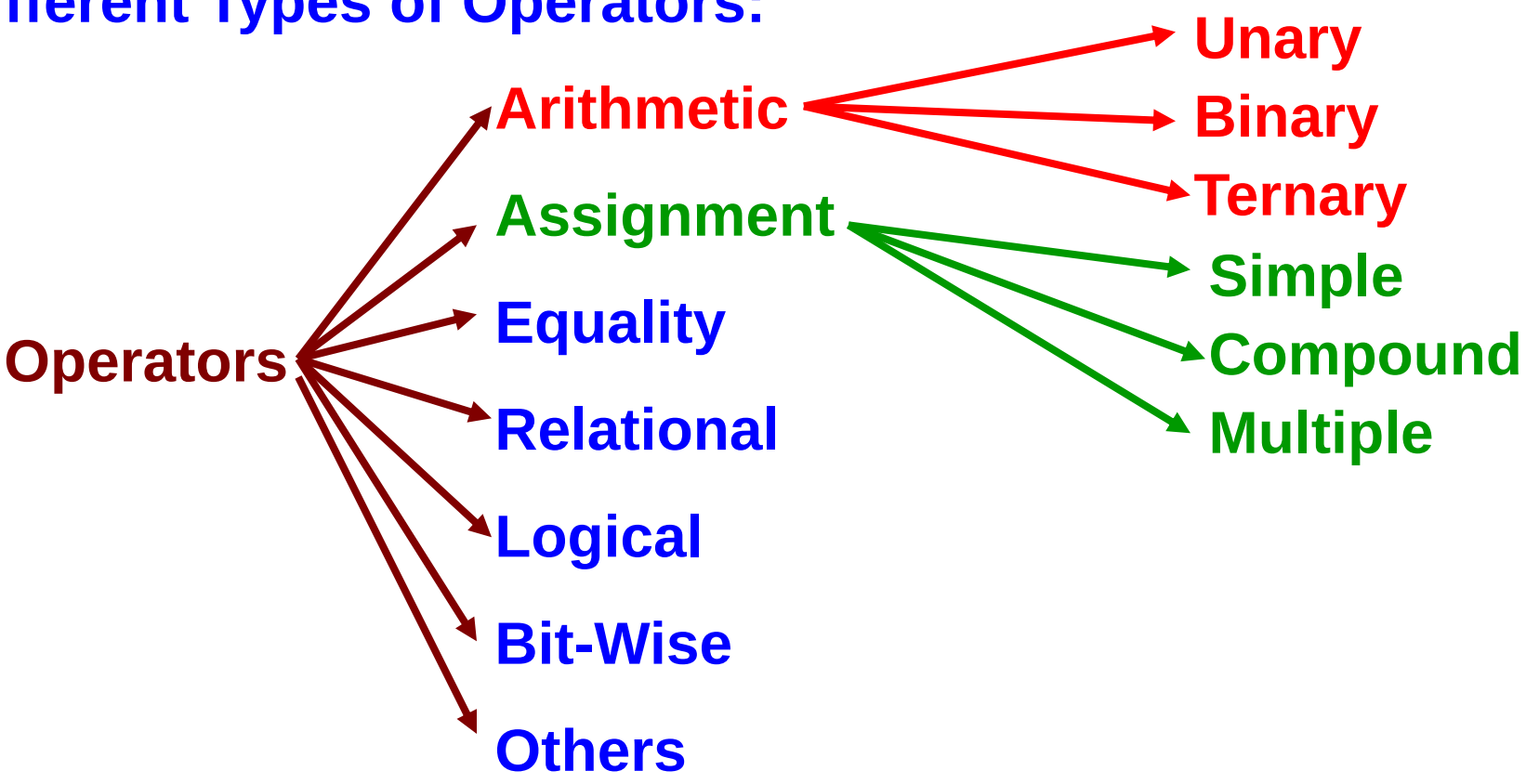
Operators in C

Operators

Operators are symbols that specify the mathematical, logical or relational operation to perform.

Different Types of Operators:



Operators in C

Arithmetic Operators in C

Operators which performs arithmetic operation on operands are called arithmetic operators.

Unary Operators:

Operate on one operand

Binary Operators:

Operate on two operands

Ternary Operators:

Operates on three operands

Operators in C

Unary Arithmetic Operator

+	Identity Operator or unary plus
-	Negation Operator or unary minus
++	Increment Operators
--	Decrement operators

Negation operator changes the sing of the operand

Identity operator does not change the operand but exist for maintaining symmetry with '+'

Increment operator is used to increment the value of a variable;

Decrement operator is used to decrement the value

++ and -- will be discussed in details

Operators in C

Binary Arithmetic Operator

+	Addition Operator	A + B
-	Subtraction Operator	A - B
*	Multiplication Operator	A * B
/	Division Operator	A / B
%	Modulus Operator	A % B

All operators except % can work on both integers and floating point numbers or mixed numbers

If the operation results in a value more than the range of the data type then the MSBs are dropped (True for unsigned data type)

For signed data type it is implementation dependent

Operators in C

Arithmetic Operators

- **If the data type of both the operands are same than operation will be performed in the same Data type and result will be in the same data type**

- **But the minimum data type should be integer (char and short are converted into integer before actual operation)**

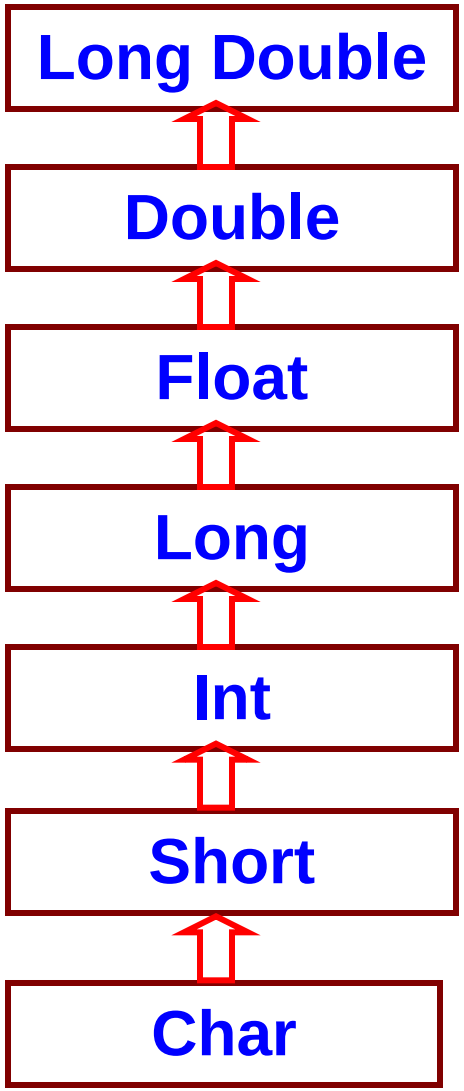
- **If they are different than the data type of one operand will be changed to match the other (called automatic type conversion) and the operation is performed**

data type of which operand will change???

- **The data type of the least comprehensive variable changes to the data type of the most comprehensive variable. Type Promotion**

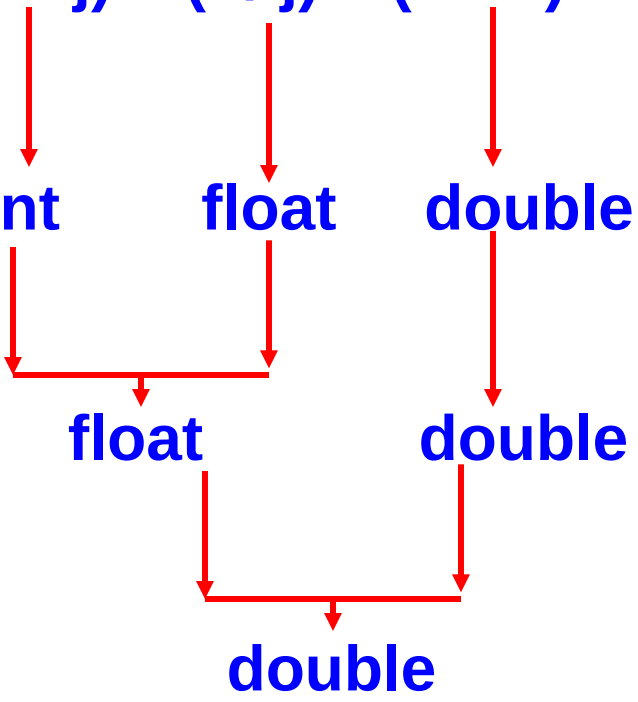
Operators in C

Automatic Type Conversion



Char c; int j; float f; double d, r;

r = (c * j) + (f / j) – (f + d)



Operators in C

Precedence in Expressions

- If there are multiple operators exists in an expression which will execute first.
- Precedence Defines the order in which an expression is evaluated

Operators in C

Precedence in Expressions – Example

```
1 + 2 * 3 - 4 / 5 =
1 + (2 * 3) - (4 / 5)
```

B.O.D.M.A.S.

B stands for brackets,
O for Order (exponents),
D for division,
M for multiplication,
A for addition, and
S for subtraction.

Operators in C

More on precedence

for arithmetic operators

1. **Unary operators has the highest precedence**
2. **`*`, `/`, `%` are at the same level of precedence**
3. **`+`, `–` are at the same level of precedence**

- **For operators at the same "level", left-to-right ordering is applied (left associativity)**

```
2 + 3 – 1 = (2 + 3) – 1 = 4
2 – 3 + 1 = (2 – 3) + 1 = 0
2 * 3 / 4 = (2 * 3) / 4 = 6 / 4
2 / 3 * 4 = (2 / 3) * 4 = 0 * 4
```

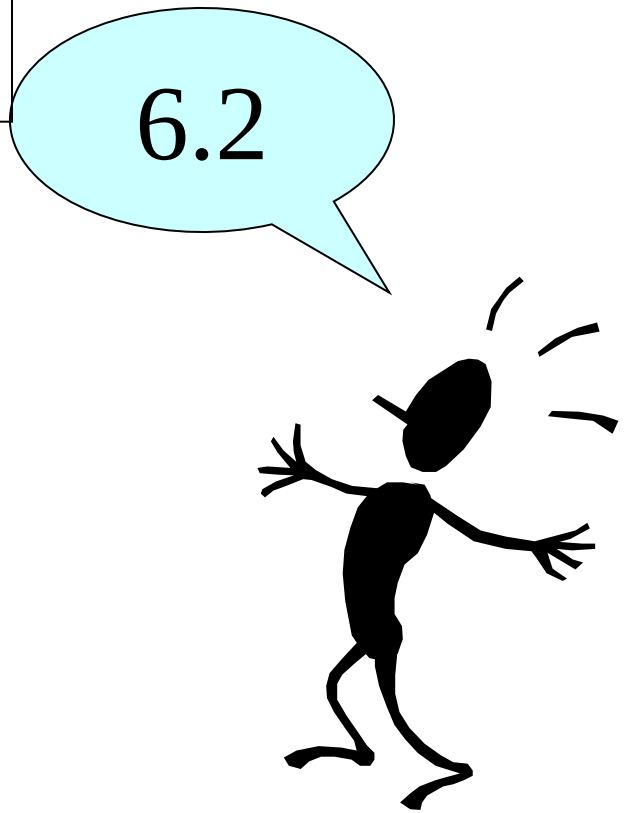
- **Parenthesis can be used to change the precedence (does not hamper speed)**

Operators in C

Precedence in Expressions – Example

```
1 + 2 * 3 - 4 / 5 =

1 + (2 * 3) - (4 / 5)
```



Operators in C

Precedence in Expressions – Example

```
1 + 2 * 3 - 4 / 5 =

1 + (2 * 3) - (4 / 5)
```

Operators in C

Precedence in Expressions – Example

```
1 + 2 * 3 - 4 / 5 =

1 + (2 * 3) - (4 / 5)
```

Integer division results in integer quotient

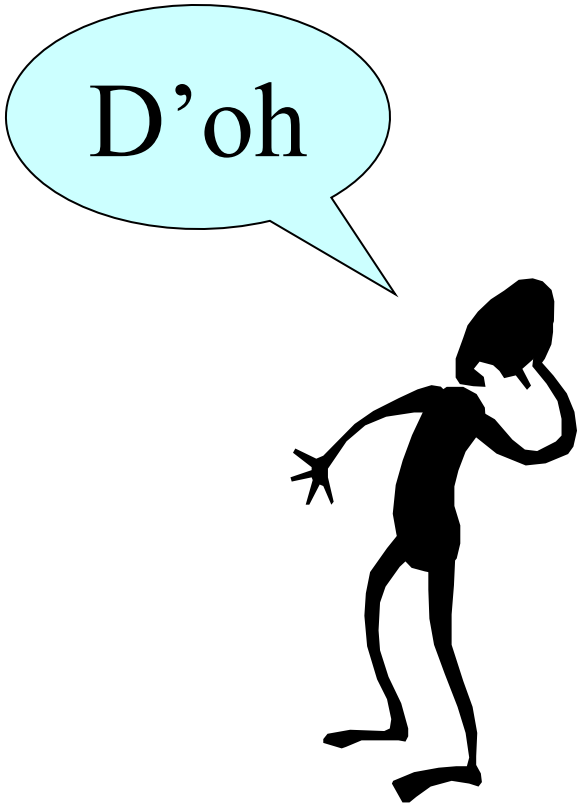
Operators in C

Precedence in Expressions – Example

```
1 + 2 * 3 - 4 / 5 =

1 + (2 * 3) - (4 / 5)
```

= 0



Operators in C

Precedence in Expressions – Example (cont)

```
1 + 2 * 3 - 4 / 5 =

1 + (2 * 3) - (4 / 5)
```

Operators in C

`int`-s and `float`-s

- `float` is a “communicable” type
- Example:

```
1 + 2 * 3 - 4.0 / 5

= 1 + (2 * 3) - (4.0 / 5)

= 1 + 6 - 0.8

= 6.2
```

Operators in C

`int`-s and `float`-s – Example 2

```
(1 + 2) * (3 - 4) / 5
= ((1 + 2) * (3 - 4)) / 5
= (3 * -1) / 5
= -3 / 5
= 0
```

Operators in C

`int`-s and `float`-s – Example 2 (cont)

```
(1 + 2.0) * (3 - 4) / 5

= ((1 + 2.0) * (3 - 4)) / 5

= (3.0 * -1) / 5

= -3.0 / 5

= -0.6
```

Operators in C

`Negative Numbers` – Example 3

```
-5/3 = ?? -5%3 = ??
```

Depends on Compiler

-5 / 3 = - 1 or - 2

-5 % 3 = - 2 or +1

Operators in C

Over flow and Under Flow Exceptions

What happens when the result of the arithmetic operation exceeds the limit of the given data type.

Does not report any error

UINT_MAX + 1 = ?? Predictable

INT_MAX + 1 = ?? Non predictable (implementation dependant)

So better to avoid the overflow and under flow while doing arithmetic operations in c

C can not handle divide by Zero Error so rises a devide by zero exception.

Operators in C

Increment (++) & Decrement (--) Operators

used to increment or decrement the value of the variable by one;

Two basic rules

1. ***operand must be a variable (can not be constants or expression)***
2. ***++ or -- can precede or succeed the operand***

Known as pre increment and post increment

int I = 5;

I ++; or ++ I (is same as I = I + 1; but executed faster)

Similarly

I --; or -- I ; (Is same as I = I -1;)

Operators in C

How pre and post increment differ

a = ++b;

increment is done first

than expression is executed

same as

b= b+1;

a = b;

a = b ++;

first expression is executed

with the old value of b than it is incremented

same as

a = b;

b = b+1;

Example-1

A = 5; C = 10;

B = ++A + C-- ;

A=?? B = ?? C=??

Examples

A = 5; C = 10;

B = A++ + C++ - --C ;

A=?? B = ?? C=??

Operators in C

Try more examples

```
int main()
{
        int A = 5, B;
        B = ++A + ++A + ++A + A;
        printf("A: %d, B:%d", A, B);
        getch();
        return 0;
}
```

Operators in C

Assignment Operators (=)

variable_name = expression;

Lvalue = Rvalue

I = 6;

*X = m * 23 + k;*

Lavlue should be a memory location which can store a value; a variable

where Rvalue can be a variable or a constant or an expression.

A + B = 5 +2;

Assignment operator has got the least precedence

Operators in C

Compound Assignment Operators (=)

x = x+5;

x += 5;

Similarly

-=, *= , /=, %= etc are different compound assignment statements

Operators in C

Multiple Assignment Operators (=)

x = y = z = 5; associativity is from right to left

z = 5 is an expression

value of the expression is 5 which is assigned to y

Value of the expression y = 5 is 5 which is assigned to x ;

Equivalent to

```
z = 5;
y = z;
x = y;
```

Operators in C

Type Conversion During Assignment

Variable = Expression;

What if data type of the variable is not equal to the data type of the expression;

An internal Automatic (Implicit) Type conversion is done.

If the Lvalue is of higher data type than it can easily accommodate the value of the expression.

But What if the case is reverse

In that case we are going to loose the higher order bit or some information

Operators in C

Data Loss During Conversion

Target Type	Expression Type	Possible Info Loss
Char	Short int (16 bits)	Higher order 8 bits
Char	Int (32 bit)	Higher order 24 bits
Int (16bits)	Long int(32 bits)	Higher order 16 bits
Int (32 bits)	Long int(32 bits)	None
Int	Float	Fractional part + possibly more
Float	Double	Precision and rounded
Double	Long Double	Precision and rounded

Operators in C

Comparison Operators

Equality Operator

== equal to operator (checks for equality)

!= not equal to operator (checks for in equality)

Don't get confused with == and =

Relational Operators

> ***Greater than operator***

< ***Less than operator***

<= ***Less than equal to operator***

>= ***Greater than equal to operator***

Operators in C

Table of Relational Operators	
Operator	**Meaning**
`A == B`	is A equal to B ?
`A < B`	is A less than B ?
`A <= B`	is A less than or equal to B ?
`A > B`	is A Greater than B ?
`A >= B`	is A Greater than or equal to B ?
`A != B`	is A not equal to B ?

All these operators take two operand and compare them. And return 1 for true and 0 for false

Operators in C

Boolean Expressions

Expression	Value	Expression	Value
25 < 25	false	25 != 25	false
25 <= 25	??	25 > 25	false
25 >= 25	true	25 = 25	??
-5 < 7	true	-305 <= 97	true

If A>B is FALSE than what is the value of A<B

Operators in C

Playing with Conditionals

- **int x=0, y=10, w=20, z, T=1, F=0;**

Find Out the value of Z after each statement

- **z = (x == 0);**
- **z = (x = 0);**
- **z = (x == 1);**
- **z = (x = 15);**
- **z = (x != 2);**
- **z = (x < 10);**
- **z = (x <= 50);**

Operators in C

- **Logical Operators**
  - ! Logical Negation **(Unary)**
  - && Logical AND Operator
  - || Logical OR Operator

! - Negation operator

!(T) → F

!(F) → T

|| - Logical OR Operator

F || F	F
F || T	T
T || F	T
T || T	T

&& - Logical AND Operator

F && F	F
F && T	F
T && F	F
T && T	T

Operators in C

Certain Rules

While evaluating the logical value of a expression a non zero value is treated as TRUE and Zero is treated as FALSE.

Logical operators are evaluated until an expression is known to be true or false

Example

int x=5, y=2, z;

Z = (x=0) && (++Y);

X=0
Y=2
Z=0

What is the value of X, Y, Z after the statements get executed.

Operators in C

Certain Rules (cont..)

If logical AND operation is being performed, the evaluation is stopped when it finds the first FALSE operand.

If Logical OR is being performed the evaluation stops if it finds the first TRUE expression.

Operators in C

Playing with Conditionals

```
int main
{
 int x=0, y=10, w=20, z, T=1, F=0;
```

Evaluate the value of Z after each statement

```
  z = ((x=y) < 10);
  z = (x==5 && y<15);
  z = (x==0 && y>5 && w==10);
  z = (x==0 || y>5 && w==20);
  z = (T && T && F && y && x);
  z = (F || ++x || w - 20 || x);
      return 0;
}
```

Operators in C

Operators in C

Operators in C

Operators in C