Operators in C++

Operators

- Operators are the symbols that tells compiler to perform certain mathematical or logical on data and variables

- Types of operator
  - Unary operators (One operand)
  - Binary operators(Two operands)
  - Ternary operators(Three operands)

Operators

- Operators are classified in to following groups:
  1. Arithmetic Operators
  2. Relational Operators
  3. Logical Operators
  4. Assignment Operator
  5. Increment and Decrement Operators
  6. Conditional Operators
  7. Bitwise Operators
  8. Special Operators
  9. Shorthand Operators

Arithmetic Operators

Operator	Meaning
+	Addition and unary plus
_	Subtraction and unary minus
*	Multiplication and Dereferencing
/	Division
%	Modulo Division(Remainder)

% operator cannot be used on floating point data.
During % the sign of answer is sign of first operand
14%-3 =2 , -14%3 = -2

Relational Operators

- Compares two quantities
- Result is either one(non zero or zero(true or false)

Operator	Meaning
<	Less than
<=	Less than equal to
>	Greater than
>=	Greater than equal to
==	Equal to
!=	Not Equal to

Used in decision control structures and loops

Logical Operators

- Gives result as zero or one(false or true)

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Truth Table(short ckt rule+ def)

Op1	Op2	AND(&&)	OR(||)	NOT(!) of op1
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Shorthand Operators

Operand	Meaning
+=	a+=b => a= a+b;
-=	a-=b => a= a-b;
*=	a*=b => a= a*b;
/=	a/=b => a= a/b;
%=	a%=b => a= a%b;
^=	a^=b => a= a^b;
&=	a&=b => a= a&b;
>>=	a>>=b => a= a>>b;
<<=	a<<=b => a= a<<b;
!=	a!=b => a= a!b;

Increment operators(unary)

- Increment(++) Prefix operator
  - ++a => a= a+1 or a+=1;
  - First 1 is added to variable the it is used in expression.
  - Ex. a=5; b= ++a;

  Here first a becomes 6 and then 6 is assigned to b

- Increment(++) Postfix operator
  - a ++ => a= a+1 or a+=1;
  - First old value is used in expression and then1 is added to variable.
  - Ex. a=5; b= a++;

  Here first 5 is assigned to b and the value of a is incremented.

Decrement operators(unary)

- Decrement(--) Prefix operator
  - --a => a= a-1 or a-=1;
  - First 1 is added to variable the it is used in expression.
  - Ex. a=5; b= --a;

  Here first a becomes 4 and then 4 is assigned to b
- Decrement(--) Postfix operator
  - a -- => a= a-1 or a--=1;
  - First old value is used in expression and then1 is added to variable.
  - Ex. a=5; b= a--;

  Here first 5 is assigned to b and the value of a is decremented.

Increment and Decrement operators(unary)

- For postfix expr is evaluated using original value and then increment /decrement happens.
- For prefix first value of of variable is incremented /decremented and then expr evaluated using changed values.
- Postfix and prefix can not be used with floats.
- Ex. int x,y =3;

  x=++y++; it's a compile time error :Lavlue required.

Conditional operator(? :)

- Ternary operator **(? :)**
- Condition **?**expr 1**:** expr 2;
- If condition is true expr 1 executes else expr 2
- Expr1 and expr2 are the statements which evaluates to certain value i.e give result
- All functions except which returns void can be used at expr1 and expr2
- General format
  - Variable = Condition **?**expr 1**:** expr 2;

Conditional operator(? :)

- Equivalent of ? :

```
if(condition true)
   variable = expr 1;
else
   variable = expr 2;
```

- Max 3 conditional operators

```
Max= a>b?(a<c?a:c):(b>c?b:c);
```

- Limitation of ? :
  - After ? Or : one statement can occur.

Bitwise Operators

- Used in device drivers programming.
- Used to manipulate data at bit level.
- Only applicable for int & char data types

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
~	One's compliment (NOT)
>>	Right shift
<<	Left Shift
^	Bitwise Ex-OR

Bitwise Operators

- -ve numbers are stored in 2's compliment form of +ve number.
- In bitwise AND,OR,Ex-Or both the operands should be of same data type(int or char)
- Truth Table for Bitwise AND,OR & Ex-OR

OP1	OP2	&	|	^
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bitwise Operators

- Complement(~) operator
  - 0 => 1
  - 1=> 0
- Right shift(>>) operator
  - Op>>n;
  - All bits shifted to right by n positions or bits.
  - Rightmost n bits will be lost and left most vacated n bits will be filled with 0
  - Equivalent to divide by 2
    - 32>>1 gives 16
- Left shift(<<) operator
  - Op<<n;
  - All bits shifted to left by n positions or bits.
  - Leftmost n bits will be lost and rightmost vacated n bits will be filled with 0
  - Equivalent to multiply by 2
    - 16<<1 gives 32

Special Operators

Operator	Meaning
,	Comma operator
Sizeof( )	Size of operator
& ,*	Pointer related operator
. ,->	Structure related operators

- Comma(,) operator
  - Used to link expressions together, expr evaluated from left to right and value of rightmost expr is total value of expr.

    EX. Value =(x=10,y=5,x+y); then value =15;

  - Left side of , is always evaluated as void ,i.e value of rightmost expr becomes value of ,separated expression.

Special Operators

- sizeof( ) operator
  - Compile time operator. i.e value is evaluated at compile time only and value it produces is treated as a constant within your program.
  - Returns no of bytes the operand occupies.
  - Operand may be variable, constant, data type qualifier.
  - Ex. m= sizeof(int); result : m=4
  - It is generally used to generate portable code that depends on size of built in datatypes.

Special Operators

- Assignment operator (=)
  - Assigns result of expression to variable.
  - Ex . v = expr; here v must be varibale
  - Ex. a+b = expr; //Lvalue required error

  Here a+b is not variable, so it can not store value.
- New operator (new)
  - Used to allocate memory dynamically
  - Ex. int *p = new int;
- Delete operator (delete)
  - Use to release/free memory dynamically
  - Ex. delete p;

Expressions

- **Expression:** collection of operands(variable and constants) connected by operators.
  - Result of expr is int if all operands are of type int. If any operand is float the result is float.
  - The result of an expr is converted in to data type of variable present on LHS of assignment operator.
  - There must be one variable on LHS of assignment operator ex. a+b = expr // not allowed.
  - No two operands can be connected directly without operator ex. 2a+3 is not allowed in expr(2*a+3)

Expression Evaluation

- **variable=expr;**
  - Expr is evaluated from left to right and then result is assigned to variable.
  - All variables in exprs must be assigned some value before evaluation.
- **How C++ evaluates expression**
  - Firstly values are assigned to all variables.
  - Execution starts from LHS to RHS.
  - Parenthesis sub expr are evaluated from left to right.
  - In nested parenthesis innermost is evaluated first.
  - Then precedence rule is used to evaluate expr.
  - Associativity rule is used if two or more operators of same precedence appear in expr.

Precedence chart

Priority	Operator	Description	Associativity
1(High)	()	Function call || bracket	LR
	[ ]	Array element reference	LR
	.	Structure member reference	LR
	->	Pointer to structure reference	LR
2	-	Unary minus	RL
	++	Increment operator	RL
	--	Decrement operator	RL
	!	Not operator	RL
	~	1's compliment	RL
	Type	Type casting	RL
	&	Address	RL
	*	Pointer derefernce	RL
	sizeof	Size of object	RL

Precedence chart

Priority	Operator	Description	Associativity
3	*	Multiplication	LR
	/	Division	LR
	%	Remainder	LR
4	+	Addition	LR
	-	Subtraction	LR
5	<<	Left shift	LR
	>>	Right shift	LR
6	<	Less than	LR
	<=	Less than equal to	LR
	>	Greater than	LR
	>=	Grater than equal to	LR
7	==	Equality	LR
	!=	Inequality	LR

Precedence chart

Priority	Operator	Description	Associativity
8	&	Bitwise AND	LR
	^	Bitwise XOR	LR
	|	Bitwise OR	LR
9	&&	Logical AND	LR
10	||	Logical OR	LR
11	? :	Conditional Operator	RL
12	=	Assignment Operator	RL
	*=, /=, %=	Shorthand Operators	RL
	&= ,^=, !=	Shorthand Operators	RL
	+= ,-=	Shorthand Operators	RL
	<<=, >>=	Shorthand Operators	RL
13(low)	,	Comma operator	RL

Rules for Expression Evaluation

- The operator having highest precedence is evaluated first.
- The operators with same precedence are evaluated using associativity.
- In case of arithmetic expression compiler always executes from Left to right

EX. a= 2 + 3 * 4

Execution

Here 3 is associated with + and * . Precedence of * is greater so fist multiplication will be done and then addition hence 2+ 12= 14

Data type promotion

- If the data type of variable on LHS of ‘=“ is different form data type of result of expr. Then result is promoted depending upon data type on LHS.
  - EX. int a=20;

    int b=3;

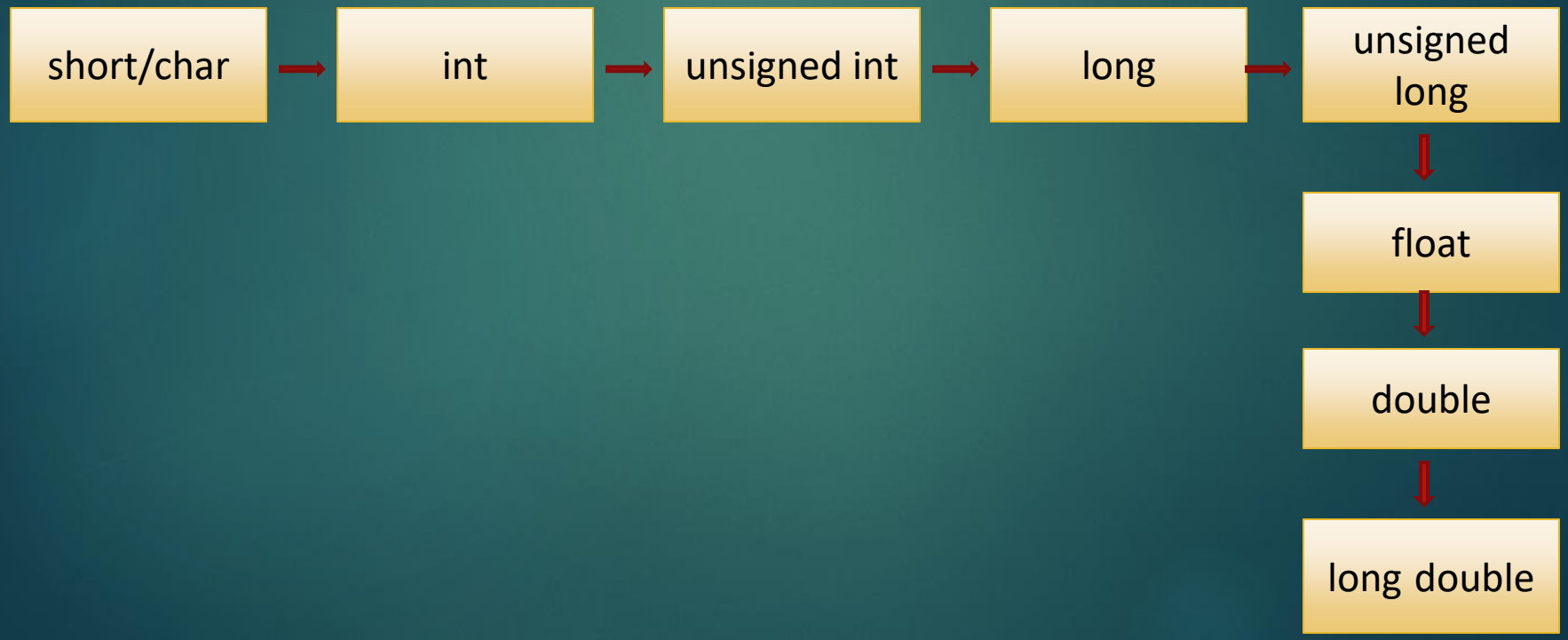
    float c=0;

    c= 20/3; => c=6.0;

  Here the int result is converted to float

Data type conversion(type cast)

- **Explicit Type Conversion(casting)**
  - (data type) exp; forced type conversion.
- **Implicit Type Conversion**
  - If a expression contains different data type variables then lower type is automatically converted to higher data type.



LValue & RValue

- **Lvalue**: Any variable whose value can be changed and it appears on LHS of Assignment operator.
- **Rvalue**: Value or variable whose value cannot be changes or expression which appears on RHS of Assignment operator.

Exercise

- Find roots of equations .
- Calculate simple interest using principal, rate of interest and period of investment.
- Check given number is odd or even number.
- Check if the entered char is small letter or capital letter.

Thank You

Operate with care!!!!

…….Operators