Data Types
Operators and Hierarchy

Data Types

- A datatype is a collection of values with shared properties
- Using types makes a program easier to read and understand
- Using types makes it easier for the compiler
- Types makes it easier to detect certain programming errors

Classes of Data Types

- Primary (Fundamental) Data Types
- User-defined Data Types
- Derived data types
- Empty Data set

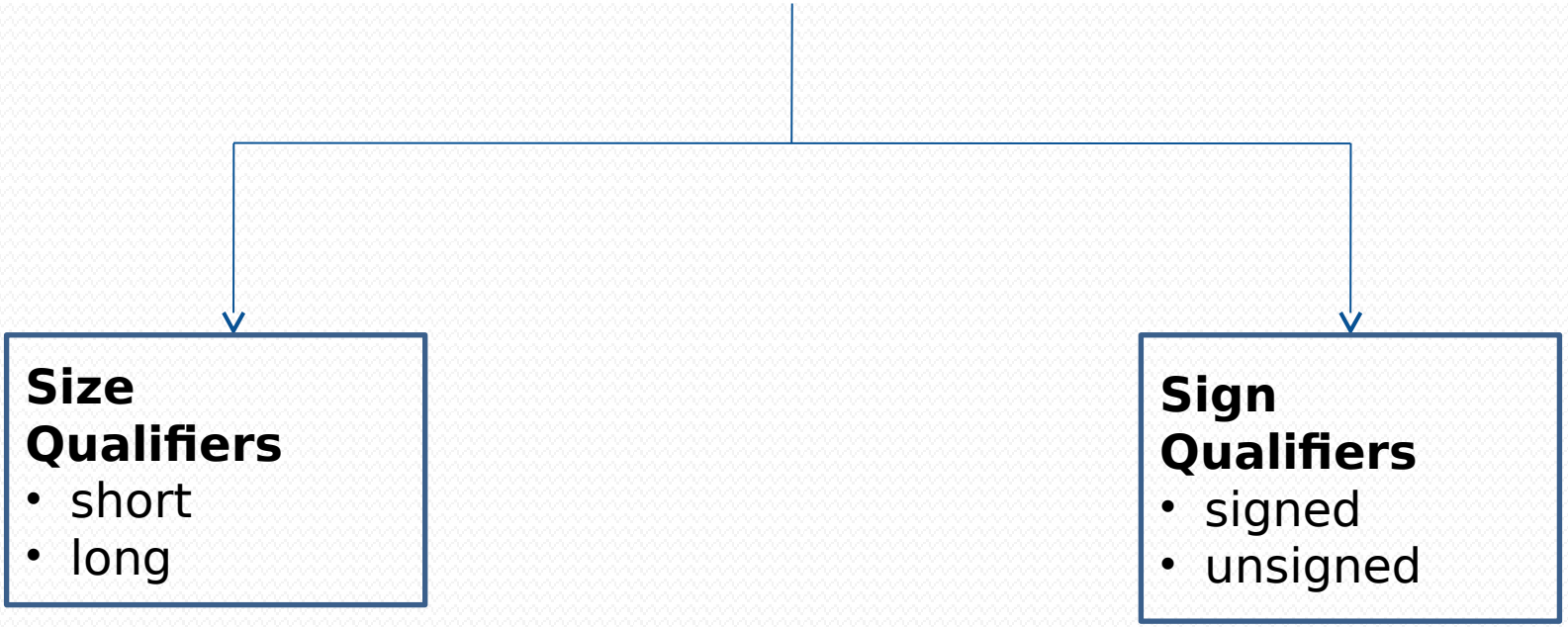
Primary Data Types

Integer	
signed type int short int long int	**unsigned type** unsigned int unsigned short int unsigned long int

Character
signed char unsigned char

Floating Point
float double long double

Use of Qualifiers



Note:

- If unsigned qualifier is assigned the number is always positive
- If signed qualifier the number may be positive or negative

Range of Data Types

Basic Data Type	Data Type with qualifiers	Size (Bytes)	Size (Bits)	Range
char	char or signed char unsigned char	1 1	8 8	-128 to 127 0 to 255
int	int or signed int unsigned int short int or signed short int unsigned short int long int or signed long int unsigned long int	2 2 1 1 4 4	16 16 8 8 32 32	-32768 to 32767 0 to 65535 -128 to 127 0 to 255
float	float	4	32	3.4e-38 to 3.4e+38
double	double long double	8 10	64 80	1.7e-308 to 1.7e+308 3.4e-4932 to 1.1e+4932

Example

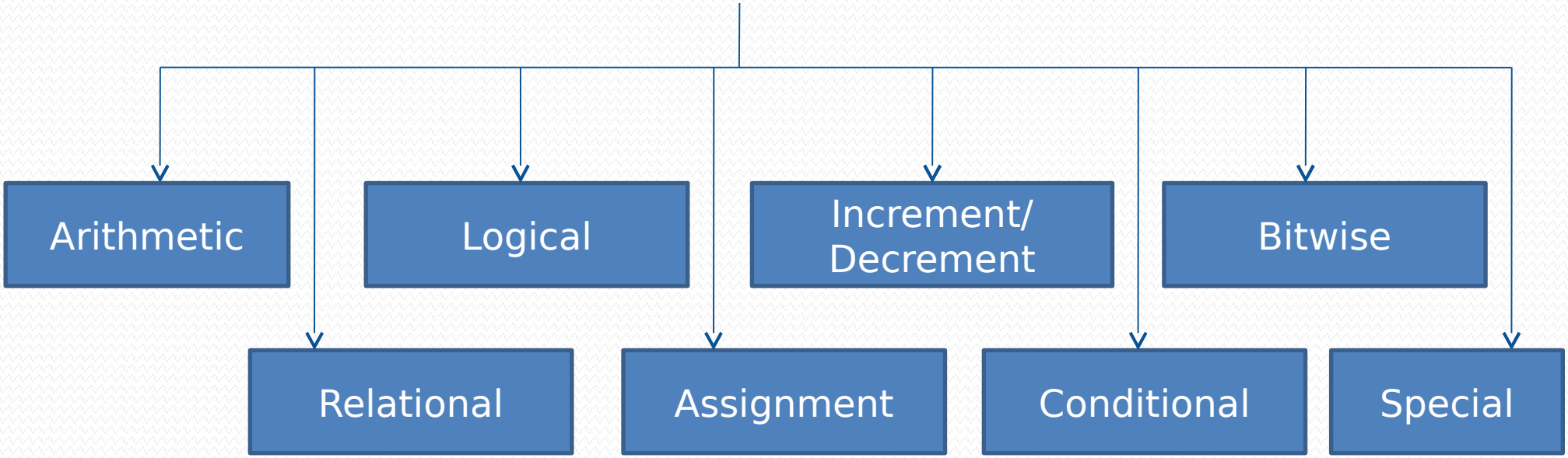
```
#include<stdio.h>
main()
{
    int a, b, c, d;          //declaration
    unsigned int u = 10;          //declaration

    a = 12; b = -24;          //assignment
    c = a + u; d = b + u;  //assignment
    printf("a+u=%d, b+u=%d\n", c, d);
}
```

output:

```
a+u=22, b+u=-14
```

Types of Operators



Arithmetic Operator

Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo Division

```
#include<stdio.h>
int main()
{
        int x;
        x=5*4+20/2-6;
        printf("%d",x);
        return 0;
}
```

- The second operand must be non-zero for division operator.
- The modulus operator can't be used with floating point number.

Logical Operators

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

Relational Operators

Operator	Meaning
<	is less than
<=	Less than or equal to
>	is greater than
>=	greater than or equal to
==	is equal to
!=	is not equal to

- Compare values of 2 expression depending on their relations
- If the relation is true it returns 1 otherwise 0
- Has highest priority than logical operator
- example 1
  - int i=10, i=20,k
  - k = i < j
- example 2
  - k = 100 == 100
  - k = 100 >= 100

Assignment Operator

- Assignment operator are used to assign the result to an expression
- “=” is the assignment operator

Statement with simple assignment operator	Statement with shorthand assignment operator
a=a+1	a+=1
a=a-1	a-=1
a=a*(n-1)	a*=n-1
a=a%b	a%=b

Increment/Decrement Operator

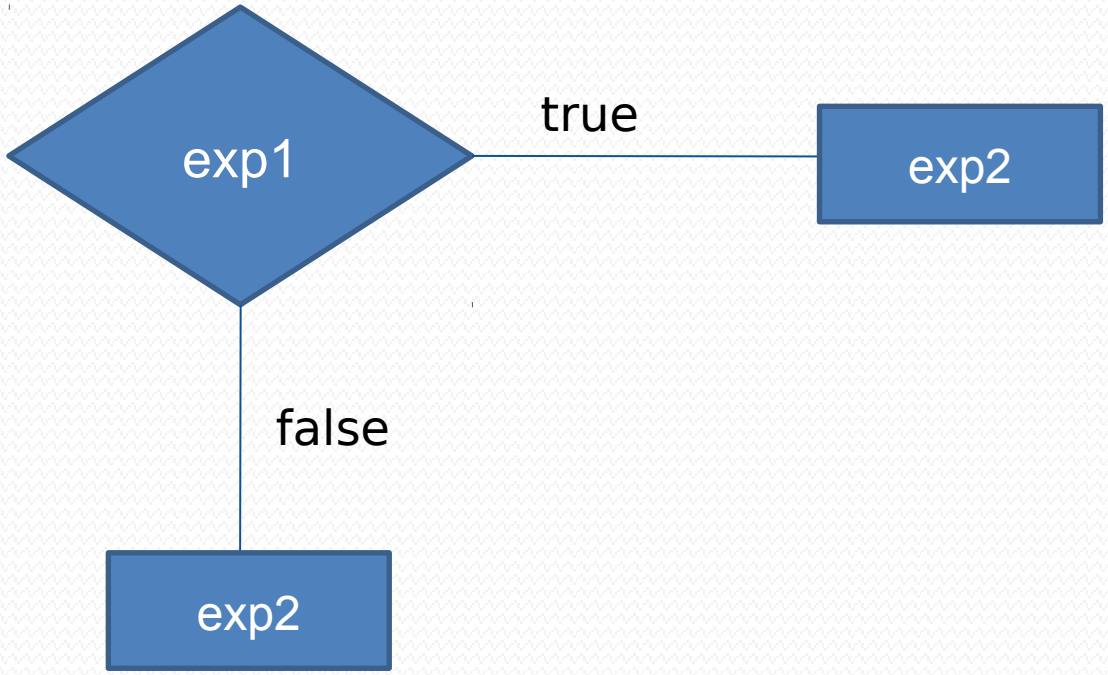
- “++” and “--” are the operators
- ++ adds one to operand while -- subtracts one
- Both are unary operators

Postfix/Prefix ++ and --

- Require variable as their operands
- Postfix ++ (or --): First the value of variable is used in the operation and then increment/decrement is performed
- Prefix ++ (or --): First the value of variable is incremented/decremented then new value is used

Conditional Operator

- Also known as ternary operator
  - exp1 ? exp2 : exp3



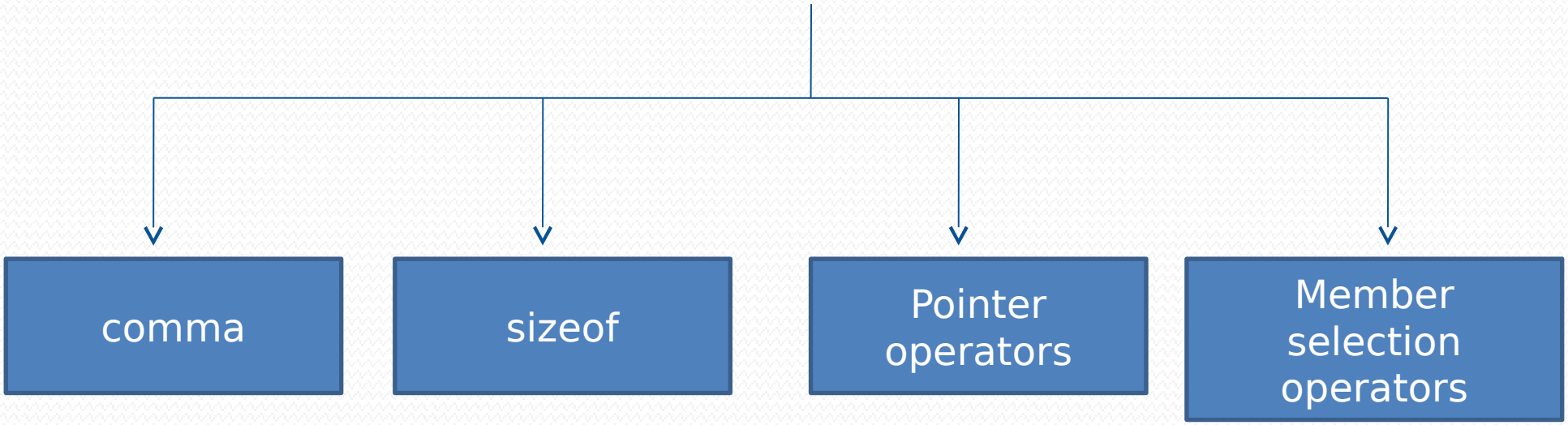

```
a=10;
b=15;
x=(a>b)?a:b;
```

Bitwise Operator

- For manipulation of data at bit level
- Bitwise operator cannot be applied to float or double

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right

Special Operators



Comma Operator

- To link the related expressions together
- Evaluated from Left to Right
- Ex:
  - value=(x=10,y=5,x+y);
  - t=x,x=y,y=t;

sizeof operator

- Returns the number of bytes the operand occupies
- The operand can be variable, constant or data type qualifier
- Ex:
  - m=sizeof(sum);
  - n=sizeof(long int);
  - k=sizeof(235L);

Precedence of Operators

- There are 2 different priorities of arithmetic expression
  - High Priority: * / %
  - Low Priority: + -
- The equation is evaluated in two passes
  - First pass: High priority operators
  - Second pass: Low priority operators

Expression: x=9-12/3+3*2-1

- 1st Pass

x=9-4+3*2-1

x=9-4+6-1

2nd Pass

x=5+6-1

x=11-1

x=10

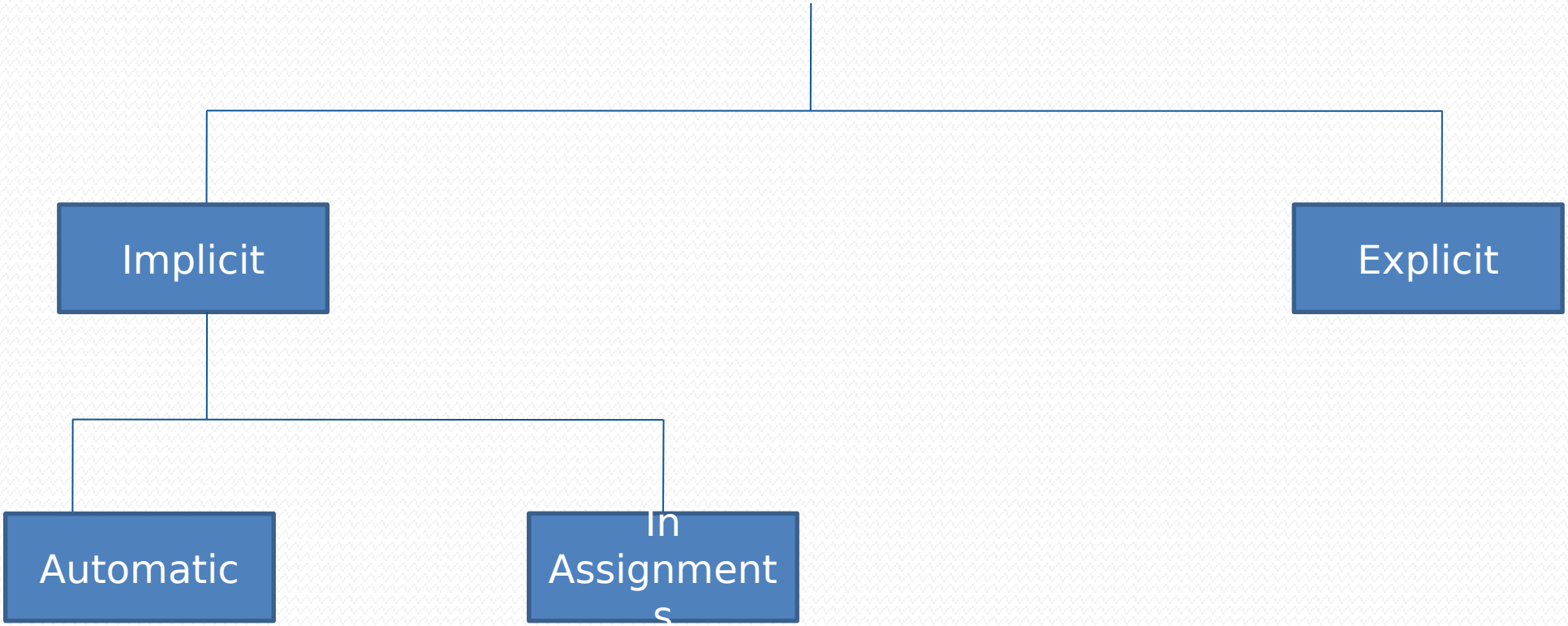
- Combines operands(variables , constants) and operators to produce new values
  - 3+x*(i+j)
- Constant expression
- Variable expression
- Rules for Evaluation of Expression
  - Parenthesized sub expression from left to right are evaluated
  - If parenthesis are nested evaluation begins with innermost braces
  - If operators of same precedence are encounter then associativity is used

Rules for Evaluation of Expression

- Parenthesized sub expression from left to right are evaluated
- If parenthesis are nested evaluation begins with innermost braces
- If operators of same precedence are encounter then associativity is used
- Arithmetic expression are evaluated from left to right

Operator	Description	Precedence	Associativity
() [] . -> ++ --	Parentheses (function call) (see Note 1) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement (see Note 2)	1	left-to-right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (change type) Dereference Address Determine size in bytes	2	right-to-left
* / %	Multiplication/division/modulus	3	left-to-right
+ -	Addition/subtraction	4	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	5	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	6	left-to-right
== !=	Relational is equal to/is not equal to	7	left-to-right
&	Bitwise AND	8	left-to-right
^	Bitwise exclusive OR	9	left-to-right
|	Bitwise inclusive OR	10	left-to-right
&&	Logical AND	11	left-to-right
||	Logical OR	12	left-to-right
?:	Ternary conditional	13	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	14	right-to-left
,	Comma (separate expressions)	15	left-to-right

Type Conversions

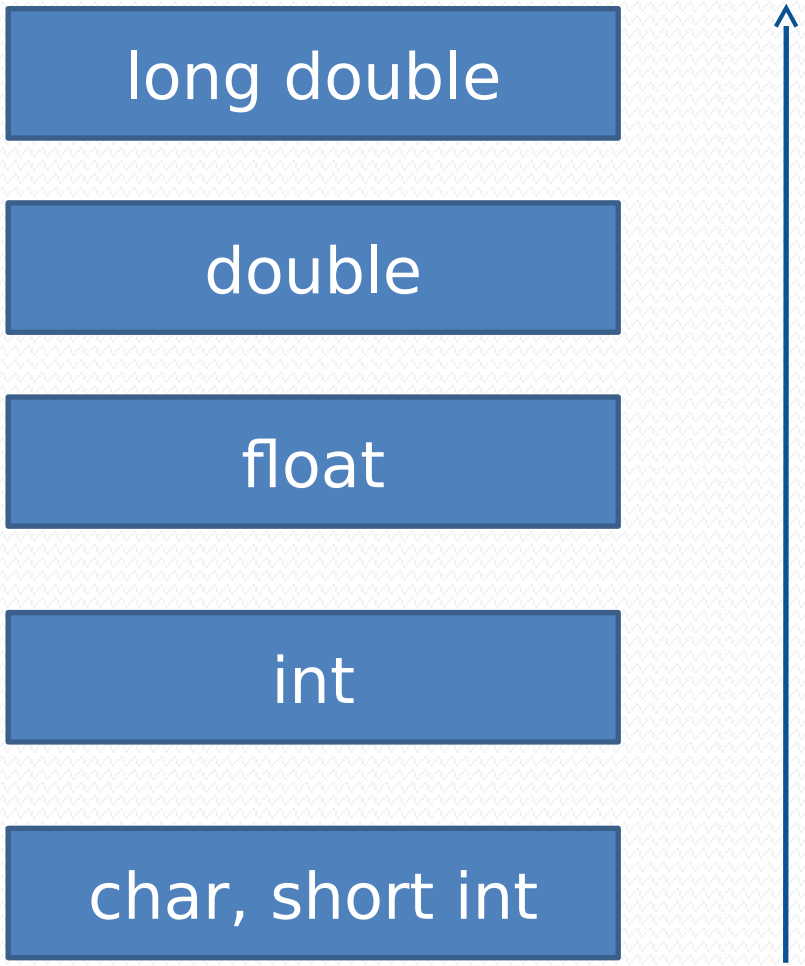


The data type of one operand is converted into data type of another operand

Implicit Type Conversion

- Implicit type conversion, also known as coercion
- An automatic type conversion by the compiler
- If operands are of different types than lower type is automatically converted to higher type

Automatic



In Assignment

- Type of right hand side is converted to type of left hand side

- If right hand operand is lower rank than it will be promoted
  - float = int
  - int = char

- If right hand operand is higher rank than it will be demoted
  - char=int
  - int=float

Explicit/Type Casting

- Is done with the help of cast operator
- Cast operator is a unary operator that is used for converting an expression to a particular data type
- Syntax:
  - (datatype) expression
- Ex:

```
int x,y;
float x=(float)x/y;
```

Example: float p=6/4; output: 1.000000

float p=(float)6/4; output: 1.500000