Akademin

C Programming

Operators & Expressions

Operators and Expressions

- An expression consists of a sequence of constants, identifiers, function calls and operators
  - Which is evaluated by performing the operations. E.g. **(12 + (x * y) / z**
- An expression can be a single constant, string literal, or the identifier of an object or function or a complex composite of them enclosed in parentheses. E.g. **((a + n/b)+(c&d))**
- Every expression has a type and it is the type of the evaluated value of the expression.
  - If an expression has no value, its type is void.
  - E.g. int a = 3; float b = 2.5f; char ch = 'a'; **(ch + a * b) / (b + a)** => its type is float
- **lvalue** and **rvalue** expressions (left expression = right expression)
- An lvalue expression is an expression which can appear on the left side of an assignment
  - E.g. int a, b = 10, c = 5; **a** = b / c; // **a** is an lvalue expression (= is the assignment operator)
  - An lvalue can always be resolved to the object's address. Exceptions: **bit-fields** and **register** variables

Operators and Expressions

- An **rvalue** is an expression appears on the right side of an assignment but not left side
  - E.g. (a +b), 123, etc. int a = **b + 123**;
- Evaluation of expressions
  - An expression containing several operators is evaluated according to
    - The **precedence** and **associativity** of the operators
      - The **precedence** of operators determines
        - The priority order of the operators and
        - Which part of the expression is treated as the operand(s) of each operator.
        - E.g. **a + b * c** is evaluated as **a + (b × c)** not **(a + b) × c**
      - The **associativity** of operators specifies if operands are grouped with operators from left to right, or from right to left.
        - E.g. a * b / c; is evaluated as **(a × b) / c** and a = b = c; **is** is evaluated as **a = (b = c)**

Operators and Expressions

❖ The precedence and associativity of operators in C

Precedence	Operators	Associativity
1	Postfix operators: **[] () . -> ++ --**	Left to right
2	Unary operators: **++ -- ! ~ + − * & sizeof**	Right to left
3	The cast operator: **(type_name)**	Right to left
4	Multiplicative operators: *** / %**	Left to right
5	Additive operators: **+ -**	Left to right
6	Shift operators: **<< >>**	Left to right
7	Relational operators: **< <= > >=**	Left to right
8	Equality operators: **== !=**	Left to right

Precedence	Operators	Associativity
9	Bitwise AND: **&**	Left to right
10	Bitwise XOR: **^**	Left to right
11	Bitwise OR: **|**	Left to right
12	Logical AND: **&&**	Left to right
13	Logical OR: **||**	Left to right
14	The conditional operator: **? :**	Right to left
15	Assignment operators: **= += -= *= /= %= &= ^= |= <<= >>=**	Right to left
16	The comma operator: **,**	Left to right

Operators and Expressions

❖ Arithmetic operators

Operator	Name	Example	Description
*****	Multiplication	x * y	The result of x multiplied by y
/	Division	x / y	The result of x divided by y. If y is zero the behavior is undefined. If y is zero and **INFINITY** has been defined by the implementation (math.h), the result is INFINITY. In math.h NAN and INFINITY may be implemented. An example of **NAN** (Not A Number): **sqrt(-1)** An example of **INFINITY**: **(1.0 / 0.0)**
%	Modulus	x % y	The remainder of x divided by y. If y is zero the behavior is undefined. Note that both the operands shall be integer numbers.
+	Addition	x + y	The sum of x and y
-	Subtraction	x - y	The difference of x and y
+ (Unary)	Positive Sign	+x	The value of x
- (Unary)	Negative Sign	-x	The arithmetic negation of x

Akademin

Operators and Expressions

❖ Comparative operators

Operator	Name	Example	Description
<	Lesser than	x < y	1 if x is lesser than y; otherwise, 0
<=	Lesser than or equal to	x <= y	1 if x is lesser than or equal to y; otherwise, 0
>	Greater than	x > y	1 if x is greater than y; otherwise, 0
>=	Greater than or equal to	x >= y	1 if x is greater than or equal to y; otherwise, 0
==	Equal to	x == y	1 if x is equal to y; otherwise, 0
!=	Not equal to	x != y	1 if x is not equal to y; otherwise, 0

Operators and Expressions

Logical Operators

Operator	Name	Example	Description
&&	Logical AND	x && y	1 if both of the operands are not equal to zero; otherwise, 0
||	Logical OR	x || y	1 if one of the operands is not equal to zero; otherwise, 0
!	Logical NOT	!x	1 if x is equal to zero; otherwise, 0

Bitwise and Shift Operators

Operator	Name	Example	Description
&	Bitwise AND	x & y	1 if both of the operands are 1; otherwise, 0
|	Bitwise OR	x | y	1 if at least one of the operands is 1; otherwise, 0
~	Bitwise NOT	~x	1 if x is zero; otherwise, 0
^	Bitwise XOR	x ^ y	1 if the operands are not equal; otherwise, 0
>>	Shift to right	x >> y	x shifted y times to right. x and y shall be of unsigned integers
<<	Shift to left	x << y	x shifted y times to left. x and y shall be of unsigned integers

Operators and Expressions

Assignment Operators

Operator	Name	Example	Description
=	Simple assignment	x = y	y is assigned to x
+=	Compound assignment x operator= y is equivalent to x = x operator (y)	x += y	Is equivalent to x = x + y
-=		x -= y	Is equivalent to x = x - y
***=**		x *= y	Is equivalent to x = x * y
/=		x /= y	Is equivalent to x = x / y
%=		x %= y	Is equivalent to x = x % y
&=		x &= y	Is equivalent to x = x & y
^=		x ^= y	Is equivalent to x = x ^ y
|=		x |= y	Is equivalent to x = x | y
<<=		x <<= y	Is equivalent to x = x << y
>>=		x >>= y	Is equivalent to x = x >> y

Operators and Expressions

Increment and decrement operators

Operator	Name	Example	Description
++	Postfix increment	x++	Is equivalent to x = x + 1; **x is changed after it is used in an expression**
	Prefix increment	++x	Is equivalent to x = x + 1; **x is changed before it is used in an expression**
--	Postfix decrement	x--	Is equivalent to x = x - 1; **x is changed after it is used in an expression**
	Prefix decrement	--x	Is equivalent to x = x - 1; **x is changed before it is used in an expression**

Memory Addressing Operators

Operator	Name	Example	Description
&	Address operator	&x	Address of x
*****	Dereference operator	*x	The object or function that x points to
[ ]	Subscript operator	x[y]	The element with the index y in the array x
.	Dot operator	x.y	The member named y in the structure or union x
->	Arrow operator	x->y	The member named y in the structure or union that x points to

Operators and Expressions

❖ Miscellaneous Operators

Operator	Name	Example	Description
( )	Parentheses operator	(x + y)	Defines expression, conditions and parameters of functions
(type) {list}	Compound literal operator	(int [2]) { 1, 2 }	Defines an unnamed object that has at ype and the values listed
sizeof	Size of operator	sizeof x	The number of bytes occupied in memory by x
_Alignof	Alignment operator	_Alignof(float)	The minimum distance between the locations of two such objects in memory. _Alignof(float) is 4 bytes
(type)	Type casting operator	(char) x	The value of x converted to the specified type (char)
? :	Ternary operator	x ? y : z	The value of y, if x is true (i.e., nonzero); otherwise, the value of z
,	Comma operator	x,y	Evaluates first x, then y; the result of the expression is the value of y

Operators and Expressions

❖ Operands shall not be of an inappropriate essential type

Operator	**Operand**	**Essential type category of arithmetic operand**					
		Boolean	**character**	**enum**	**signed**	**unsigned**	**floating**
[ ]	integer	×	×				×
+ (unary)		×	×	×			
- (unary)		×	×	×		×	
+ -	either	×		×			
* /	either	×	×	×			
%	either	×	×	×			×
< > <= >=	either	×					
== !=							

Operators and Expressions

❖ Operands shall not be of an inappropriate essential type

Operator	Operand	Essential type category of arithmetic operand					
		Boolean	character	enum	signed	unsigned	floating
! && ||	any		×	×	×	×	×
<< >>	left	×	×	×	×		×
<< >>	right	×	×	×	×		×
~ & | ^	any	×	×	×	×		×
?:	1st		×	×	×	×	×
?:	2nd and 3rd						

➢ Expressions of essentially character type shall not be used inappropriately in addition and subtraction

- Exceptions: convert between digits in the range ‘0’ to ‘9’ and the corresponding ordinal value
- Convert a character from lowercase to uppercase and vice versa

Akademin