S5_2 Operators

Type Conversions in Expressions

- C permits mixing of constants and variables of different types in an expression
- C automatically converts any intermediate values to the proper type so that the expression can be evaluated without losing any signification
- This automatic conversion is known as implicit type conversion

Type Conversions in Expressions

- The final result of an expression is converted to the type of the variable on the left of the assignment sign before assigning the value to it

- However the following changes are introduced during the final assignment
  - Float to int causes truncation of the fractional part
  - Double to float caused rounding of digits
  - Long int to int causes dropping of the excess higher order bits

Type Conversions in Expressions

- **Explicit type conversion**
  - **There are instances when we want to force a type conversion in a way that is different from the automatic conversion**

    E.g ratio=57/67

  - **Since 57 and 67 are integers in the program , the decimal part of the result of the division would be lost and ratio would represent a wrong figure**

  - **This problem can be solved by converting locally as one of the variables to the floating point as shown below:**

    The general form of a cast is

    - **(type-name) expression**

    - **Eg: ratio= (float) 57/67**

Type Conversions in Expressions

- The operator (float) converts the 57 to floating point then using the rule of automatic conversion
- The division is performed in floating point mode, thus retaining the fractional part of result
- The process of such a local conversion is known as explicit conversion or casting a value

The Type Cast Operator

int a =150;

float f; f = (float) a / 100; // type cast operator

- **The type cast operator has the effect of converting the value of the variable 'a' to type float for the purpose of evaluation of the expression.**
- **This operator does NOT permanently affect the value of the variable 'a';**
- **The type cast operator has a higher precedence than all the arithmetic operators except the unary minus and unary plus.**
- **Examples of the use of type cast operator:**

  (int) 29.55 + (int) 21.99 results in 29 + 21

  (float) 6 / (float) 4 results in 1.5

  (float) 6 / 4 results in 1.5

Type Conversions in Expressions

Example	Action
x=(int) 7.5	7.5 is converted to integer by truncation
a=(int) 21.3/(int)4.5	Evaluated as 21/4 and the result would be 5
b=(double)sum/n	Division is done in floating point mode
y=(int)(a+b)	The result of a+b is converted to integer
z=(int)a+b	a is converted to integer and then added to b
p=cos((double)x)	Converts x to double before using it

Integer and Floating-Point Conversions

- Assign an integer value to a floating variable: does not cause any change in the value of the number; the value is simply converted by the system and stored in the floating format.
- Assign a floating-point value to an integer variable: the decimal portion of the number gets truncated.
- Integer arithmetic (division):
  - int divided to int => result is integer division
  - int divided to float or float divided to int => result is real division (floating-point)

Integer and Floating-Point Conversions

```
#include <stdio.h>
int main ()
{           float f1 = 123.125, f2;
            int i1, i2 = -150;
            i1 = f1; // float to integer conversion
            printf ("float assigned to int produces");
            printf("%d\n",i1);
            f2 = i2; // integer to float conversion
            printf("integer assigned to float produces");
            printf("%d\n",f2);
            printf("integer assigned to float produces");
            printf("%f\n",f2);
            i1 = i2 / 100; // integer divided by integer
            printf("integer divided  by 100 produces");
            printf("%d\n",i1);
            f1 = i2 / 100.0; // integer divided by a float
            printf("integer divided  by 100.0 produces");
            printf("%f\n",f1);
            return 0;
}
```

123

0

-150.0

-1

-1.500

The assignment operators

- The C language permits you to join the arithmetic operators with the assignment operator using the following general format: op=, where op is an arithmetic operator, including +, –, *, /, and %.

- Example:

  count += 10;

  - Equivalent to:

  count=count+10;

- Example: precedence of op=:

  a /= b + c

  - Equivalent to:

  a = a / (b + c)

The conditional operator (? :)

condition* ? *expression1* : *expression2

- *condition* is an expression that is evaluated first.
- If the result of the evaluation of *condition* is TRUE (nonzero), then *expression1* is evaluated and the result of the evaluation becomes the result of the operation.
- If *condition* is FALSE (zero), then *expression2* is evaluated and its result becomes the result of the operation.

maxValue = (a > b) ? a : b;

Equivalent to:

```
if ( a > b )
        maxValue = a;
else
        maxValue = b;
```

Comma (,)operator

- **The coma operator is used basically to separate expressions.**

  i = 0, j = 10; // in initialization [l → r]

- **The meaning of the comma operator in the general expression e1, e2 is**

  "evaluate the sub expression *e1*, then evaluate *e2*; the value of the expression is the value of *e2*".

Operator precedence & Associativity

Operator Category	Operators	Associativity
Unary operators	+ – + + – – ~ !	R→L
Arithmetic operators	* / %	L→R
Arithmetic operators	+ –	L→R
Bitwise shift left	<< >>	L→R
Bitwise shift right		
Relational operators	< <= > >=	L→R
Equality operators	== !=	L→R
Bitwise AND, XOR, OR	& ^ |	L→R
Logical and	&&	L→R
Logical or	||	L→R
Assignment operator	= += – =	R→L
	*= /= %=	

Summary of Operators

Precedence	Operator	Description	Associativity
1 highest	::	Scope resolution	None
2	++ -- () [] . ->	Suffix increment Suffix decrement Parentheses (Function call) Brackets (Array subscripting) Element selection by reference Element selection through pointer	Left-to-right
3	++ -- + - ! ~ (*type*) * & sizeof	Prefix increment Prefix decrement Unary plus Unary minus Logical NOT Bitwise NOT (One's Complement) Type cast Indirection (dereference) Address-of Size-of	Right-to-left
4	.* ->*	Pointer to member Pointer to member	Left-to-right
5	* / %	Multiplication Division Modulo (remainder)	Left-to-right
6	+ -	Addition Subtraction	Left-to-right
7	<< >>	Bitwise left shift Bitwise right shift	Left-to-right
8	< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left-to-right
9	== !=	Equal to Not equal to	Left-to-right
10	&	Bitwise AND	Left-to-right
11	^	Bitwise XOR (exclusive or)	Left-to-right
12	|	Bitwise OR (inclusive or)	Left-to-right
13	&&	Logical AND	Left-to-right
14	||	Logical OR	Left-to-right
15	?:	Ternary conditional	Right-to-left
16	= += -= *= /= %= <<= >>= &= ^= |=	Direct assignment Assignment by sum Assignment by difference Assignment by product Assignment by quotient Assignment by remainder Assignment by bitwise left shift Assignment by bitwise right shift Assignment by bitwise AND Assignment by bitwise XOR Assignment by bitwise OR	Right-to-left
17 lowest	,	Comma	Left-to-right

Detailed Precedence Table

Example:

Show all the steps how the following expression is evaluated. Consider the initial values of i=8, j=5.

2*((i/5)+(4*(j-3))%(i+j-2))

Example solution:

2*((i/5)+(4*(j-3))%(i+j-2)) **i→8, j→5**

2*((8/5)+(4*(5-3))%(8+5-2))

2*(1+(4*2)%11)

2*(1+8%11)

2*(1+8)

2*9

18

Operator precedence & Associativity

Ex: (x==10 + 15 && y < 10)

Assume x=20 and y=5

Evaluation:

+ (x==25 && y< 10)

< (x==25 && true)

== (False && true)

&& (False)

Tutorial Problems

- Suppose that a=2, b=3 and c=6, What is the answer for the following: (a==5)
  (a * b > =c)
  (b+4 > a *c)
  ((b=2)==a)
- Evaluate the following:
  1. ((5 == 5) && (3 > 6))
  2. ((5 == 5) || (3 > 6))
  3. 7==5 ? 4 : 3
  4. 7==5+2 ? 4 : 3
  5. 5>3 ? a : b
  6. K = (num > 5 ? (num <= 10 ? 100 : 200) : 500); where num =30
- In b=6.6/a+(2*a+(3*c)/a*d)/(2/n); which operation will be performed first.
- If a is an integer variable, a=5/2; will return a value
- The expression, a=7/22*(3.14+2)*3/5; evaluates to
- If a is an Integer, the expression a = 30 * 1000 + 2768; evaluates to

S u m m a r y

- **We have learnt about**
  - **Type conversions**
  - **Assignment Operators**
  - **Conditional Operator**
  - **Comma Operator**
  - **Operator Precedence and Associativity**

Poll Question

Go to chat box/posts for the link to the Poll question

Submit your solution in next 2 minutes

Click the result button to view your score