References : 1. [illegible]
2. Ashok N. Kamtha

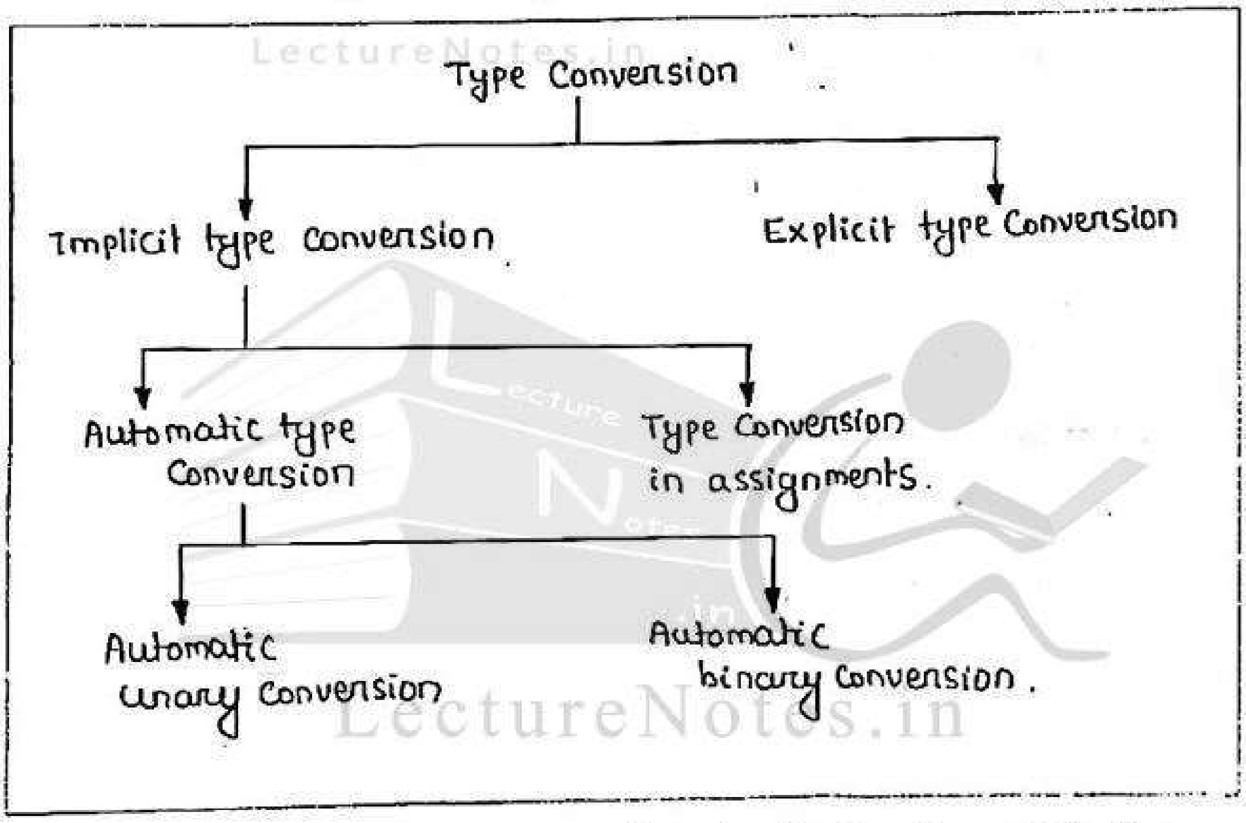
Lesson Number : 16

Type Conversion :

→ C provides the facility of mixing different types of variables and constants in an expression.

→ In these types of operations datatype of one operand is converted into data type of another operand. This is known as type conversion.

→ The different types of type conversion are :-


Type Conversion
Implicit type conversion
Explicit type Conversion
Automatic type Conversion
Type Conversion in assignments.
Automatic unary conversion
Automatic binary Conversion.

→ Implicit type conversions are done by the compiler while the explicit type conversions are user defined conversions.

1) Implicit type conversions:

These conversions are done by the C compiler according to some predefined rules of C language. The two types of implicit type conversions are automatic type conversions and type conversion in assignments.

a) Automatic type conversions :

i) Automatic unary conversions :

→ In automatic unary conversions, all operands of type char and short will be converted to int before any operation.

→ Some compilers converts all float operands to double before any operation.

ii) Automatic binary conversions :

The rules for automatic binary conversions are as -

1) If one operand is long double, then the other will be converted to long double and the result will be long double.

2) Otherwise, if one operand is double, then the other will be converted to double and the result will be double.

3) Otherwise, if one operand is float, the other will be converted to float and the result will be float.

4) Otherwise, if one operand is unsigned long int, then other will be converted to unsigned long int and the result will be unsigned long int.

5) Otherwise, if one operand is long int and other is unsigned int then ;

i) if long int can represent all the values of an unsigned int, then unsigned int will be converted to long int and the result will be long int.

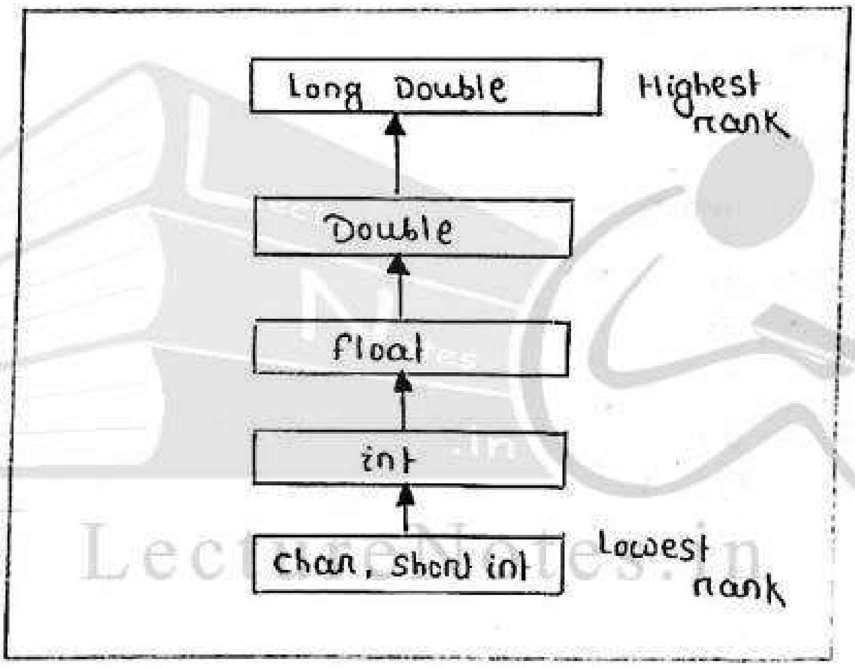
ii) Else, both the operand will be converted to unsigned long int and the result will be unsigned long int.

6) Otherwise, if one operand is long int, then the other will be converted to long int and the result will be long int.

7) otherwise, if one operand is unsigned int, then the other will be converted to unsigned int and the result will be unsigned int.

8) Otherwise, both the operand will be int and the result will be int.

If we leave aside unsigned variables, then these rules are rather simple and can be summarized by assigning a rank to each data type. Whenever, there are two operands of different data types the operand with a lower rank will be converted to the higher rank operand. This is called promotion of data type.



b) Type Conversion in assignment :

→ If the types of the two operands in an assignment expression are different, then the type of the right hand side operand is converted to the type of left hand operand.

→ Here if the right hand operand is of lower rank then it will be promoted to the left hand operand, and if it is of higher rank then it will demoted to the rank of left hand operand.

→ Some Consequences of these promotions and demotions are -

1) Some high order bits may be dropped when long is converted to int, or int is converted to short int or char.

2) Fractional part may be truncated during conversion of float type to int type.

3) when double type is converted to float type, digits are rounded off.

4) when a signed type is changed to unsigned type, the sign may be dropped.

5) when an int is converted to float, or float to double there will be no increase in accuracy or precision.

Example : /* program to understand the type conversion in assignment */

```
#include <stdio.h>
#include <conio.h>
void main()
{
 char c1, c2 ;
 int i1, i2;
 float f1, f2 ;
 c1 = 'H' ;
 i1 = 80.56 ; /* Demotion : float is converted to int, only
                80 assigned to i1 */
 f1 = 12.6 ;
c2 = i1 ; /* Demotion : int converted to char */
i2 = f1 ; /* Demotion : float converted to int */
printf ("c2 = %c , i2 = %d\n", c2, i2);
f2 = i1 ; /* promotion : int converted to float */
i2 = c1 ; /* promotion : char converted to int */
printf (" f2 = %f , i2 = %d\n", f2, i2);
getch();
}
```

Output : c2 = P i2 = 12 [A = 65 , a = 97]

f2 = 80.00 i2 = 72

2> <u>Explicit type conversion or type casting :</u>

→ There may be certain situations where implicit conversions may not solve our purpose.

→ For example ;
```
float z ;
int x = 20 , y = 3 ;
z = x/y ;
```

The value of z will be 6.0 instead of 6.66.

→ In these type of cases, we can specify our own conversions known as <u>type casting</u> <u>or</u> <u>coercion</u>. This is done with the help of <u>cast operator</u>.

→ The cast operator is a unary operator that is used for converting an expression to a particular data type temporarily. This expression can be any constant or variable.

→ The syntax of cast operator is ;

$\boxed{\text{(datatype) Expression ;}}$

Here the data type along with the parentheses is called the cast operator.

So if we write the above statement as;

z = (float) x/y ;

Now the value of z will come out be 6.66. This happen because the cast operator (float) temporarily converted the int variable x into float type and so floating point arithmetic took place and fractional part was not lost.

→ The cast operator changes the data type of variable x only temporarily for the evaluation of this expression, everywhere else in the program it will be an int variable only.

Example : /* program to illustrate the use of cast operator */

```
#include<stdio.h>
#include <conio.h>
void main()
{
  int x=5, y=2;
  float p,q;
  p= x/y;
printf("p= %f\n", p);
   q= (float)x/y ;
printf("q= %f\n", q);
getch();
}
```

output :

```
p = 2.000000
q = 2.500000
```

Initially the expression x/y is evaluated, both x and y are integers so according to integer arithmetic after division, decimal value is truncated and result is integer value 2.

This value will be assigned to p, but p is a float variable, so according to implicit type conversion in assignment the integer value 2 will be converted to float and then assigned to p. So finally the value of p is 2.0.

→ when cast operator is used, floating point arithmetic is performed, hence the value of q is 2.5.

→ Here are some other examples of usage of cast operator -

• (int) 20.3	constant 20.3 is converted to integer type and fractional part is lost (Result is 20).
• (float) 20/3	constant 20 is converted to float type, and then divided by 3 (Result is 6.66).
• (float) (20/3)	First 20 divided by 3 and then result of whole expression converted to float type (Result 6.00).
• (double) (x+y-z)	Result of expression x+y-z is converted to double.
• (double) x+y-z	First x is converted to double and then used in expression.

Assignment : what is type conversion ? what is the difference between implicit type conversion and explicit type conversion ?

→ Type conversion is a method in which the data type of one operand is converted into data type of another operand .

→ The implicit type conversions are done by the compiler .
For example here (automatic unary conversion) all operands of type char and short will be converted to int before any operation .

→ In explicit type conversions users are defined the conversions i.e. the explicit type conversions are user defined conversions .

Assignment : what is the o/p of the following program ?

```
void main()
 {
   char c1 , c2 ;
   int i1 , i2 ;
   float f1 , f2 ;
   c1 = 'H';
   i1 = 80.00 ;
   f1 = 13.6 ;
  c2 = i1 ;
  i2 = f1 ;
  f2 = i1 ;
  i2 = c1 ;
printf ( " c2 = %c  i2 = %d\n", c2, i2);
printf ("f2 = %f  i2 = %d\n", f2, i2);
}
```

output :
```
c2 = P   i2 = 13
f2 = 80.00  i2 = 72
```

Assignment : what is cast operator ?

→ Type casting is done with the help of cast operator.

→ The cast operator is a unary operator that is used for converting an expression to a particular data type temporarily. This expression can be any constant or variable.

→ The Syntax of cast operator is ;

```
(datatype) Expression ;
```

→ Example : (float) x/y ;

Assignment : what is the output of the following program ?

```
void main ( )
 {
   int x = 10 , y = 3 ;
   float p, q ;
   p = x/y ;
  printf ( " p = %f \n", p) ;
   q = (float) x/y ;
  printf ( "q = %f \n", q) ;
  getch() ;
 }
```

output :
```
p = 3.000000
q = 3.33333
```