# {Computer Programming Using C}

## Branch:CSE/IT

## Topic: Expression and Operator

BY

Prashant  Singh Yadav (Lecturer CSE)
Navneet Solanki (Lecturer IT)

# Expressions and Operators

- Examples:

```
3 + 5;
x;
x=0;
x=x+1;
printf("%d",x);
```

- Two types:
  - Function calls
  - The expressions formed by data and operators
- An expression in C usually has a value
  - except for the function call that returns void.

# Arithmetic Operators

| Operator | Symbol | Action | Example |
|---|---|---|---|
| Addition | + | Adds operands | x + y |
| Subtraction | - | Subs second from first | x - y |
| Negation | - | Negates operand | -x |
| Multiplication | * | Multiplies operands | x * y |
| Division | / | Divides first by second (integer quotient) | x / y |
| Modulus | % | Remainder of divide op | x % y |

# Assignment Operator

- x=3
  - = is an operator
  - The value of this expression is 3
  - = operator has a side effect -- assign 3 to x
- The assignment operator =
  - The side-effect is to assign the value of the right hand side (rhs) to the left hand side (lhs).
  - The value is the value of the rhs.
- For example:

```
x = ( y = 3 ) +1;    /* y is assigned 3 */
                     /* the value of (y=3) is 3 */
                     /* x is assigned 4 */
```

# Compound Assignment Operator

- Often we use “update” forms of operators
  - x=x+1,  x=x*2, ...
- C offers a short form for this:
  - Generic Form

    variable op=  expr      equivalent to  variable = variable op expr

| Operator | Equivalent to: |
|---|---|
| x *= y | x = x * y |
| y -= z + 1 | y = y - (z + 1) |
| a /= b | a = a / b |
| x += y / 8 | x = x + (y / 8) |
| y %= 3 | y = y % 3 |

  - Update forms have value equal to the final value of expression
    - i.e.,   x=3; y= (x+=3);     /* x and y both get value 6 */

# Increment and Decrement

- Other operators with side effects are the pre- and post-increment and decrement operators.
  - Increment: ++ ++x, x++
    - ++x is the same as : (x = x + 1)
      - Has value $x_{old}$ +1
      - Has side-effect of incrementing x
    - x++
      - Has value $x_{old}$
      - Has side-effect of incrementing x
  - Decrement -- --x, x--
    - similar to ++

# Relational Operators

- Relational operators allow you to compare variables.
  - They return a 1 value for true and a 0 for false.

| Operator | Symbol | Example |
|---|---|---|
| Equals | == | x == y  NOT x = y |
| Greater than | > | x > y |
| Less than | < | x < y |
| Greater/equals | >= | x >= y |
| Less than/equals | <= | x <= y |
| Not equal | != | x != y |

- There is no bool type in C.  Instead, C uses:
  - 0 as false
  - Non-zero integer as true

# Logical Operators

- &&     AND
- ||       OR
- !        NOT

!((a>1)&&(a<10))||((a<-1)&&(a>-10))

# Operating on Bits (1)

- C allows you to operate on the bit representations of integer variables.
  - Generally called bit-wise operators.
- All integers can be thought of in binary form.
  - For example, suppose ints have 16-bits
    - $65520_{10} = 1111\ 1111\ 1111\ 0000_2 = FFF0_{16} = 177760_8$
- In C, hexadecimal literals begin with 0x, and octal literals begin with 0.
    - x=65520; base 10
    - x=0xfff0; base 16 (hex)
    - x=0177760; base 8 (octal)

# Operating on Bits (2)

Bitwise operators

- The shift operator:
  - x << n
    - Shifts the bits in x n positions to the left, shifting in zeros on the right.
    - If x = $1111\ 1111\ 1111\ 0000_2$

      x << 1 equals $1111\ 1111\ 1110\ 0000_2$
  - x >> n
    - Shifts the bits in x n positions right.
      - shifts in the sign if it is a signed integer (arithmetic shift)
      - shifts in 0 if it is an unsigned integer
    - x >> 1 is $0111\ 1111\ 1111\ 1000_2$ (unsigned)
    - x >> 1 is $1111\ 1111\ 1111\ 1000_2$ (signed)

# Operating on Bits (3)

- Bitwise logical operations
  - Work on all integer types
    - & Bitwise AND

      x= 0xFFF0

      y= 0x002F

      x&y= 0x0020
    - | Bitwise Inclusive OR

      x|y= 0xFFFF
    - ^ Bitwise Exclusive OR

      x^y= 0xFFDF
    - ~ The complement operator

      ~ y= 0xFFD0

      » Complements all of the bits of X

# Shift, Multiplication and Division

- Multiplication and division is often slower than shift.
- Multiplying 2 can be replaced by shifting 1 bit to the left.

```
n = 10
printf("%d = %d" , n*2, n<<1);
printf("%d = %d", n*4, n<<2);
……
```

- Division by 2 can be replace by shifting 1 bit to the right.

```
n = 10
printf("%d = %d" , n/2, n>>1);
printf("%d = %d", n/4, n>>2);
```

# Operator Precedence

| Operator | Precedence level |
|---|---|
| ( ) | 1 |
| ~, ++, --, unary - | 2 |
| *, /, % | 3 |
| +, - | 4 |
| <<, >> | 5 |
| <, <=, >, >= | 6 |
| ==, != | 7 |
| & | 8 |
| ^ | 9 |
| \| | 10 |
| && | 11 |
| \|\| | 12 |
| =, +=, -=, etc. | 14 |

- We'll be adding more to this list later on...

# An Example

- What is the difference between the two lines of output?

```
#include <stdio.h>
int main ()
{
  int w=10,x=20,y=30,z=40;
  int temp1, temp2;
  temp1 = x * x /++y + z / y;
  printf ("temp1= %d;\nw= %d;\nx= %d;\ny= %d;\nz= %d\n",
            temp1, w,x,y,z);
  y=30;
  temp2 = x * x /y++ + z / y;
  printf ("temp2= %d;\nw= %d;\nx= %d;\ny= %d;\nz= %d\n",
            temp2, w,x,y,z);
  return 0;
}
```

# Conditional Operator

- The conditional operator essentially allows you to embed an "if" statement into an expression
- Generic Form

  exp1 ? exp2 : exp3

  if exp1 is true (non-zero)
  value is exp2
  (exp3 is not evaluated)
  if exp1 is false (0),
  value is exp3
  (exp2 is not evaluated)

- Example:

  ```
  z = (x > y) ? x : y;
  ```

  - This is equivalent to:

  ```
  if (x > y)
     z = x;
  else
     z = y;
  ```

# Comma Operator

- An expression can be composed of multiple subexpressions separated by commas.
  – Subexpressions are evaluated left to right.
  – The entire expression evaluates to the value of the *rightmost subexpression*.
- Example:

```
x = (a++, b++);
```

    - a is incremented
    - b is assigned to x
    - b is incremented

  – Parenthesis are required because the comma operator has a lower precedence than the assignment operator!
- The comma operator is often used in for loops.

# Comma Operator and For Loop

- **Example:**
- `int i, sum;`
- `for (i=0,sum=0;i<100;i++){`
- `    sum += i;`
- `}`
- `printf("1+...+100 = %d", sum);`