Assignment Statement

- Used to assign values to variables, using the assignment operator (=).
- General syntax:

  variable_name = expression;

- Examples:

```
velocity = 20;
b = 15;  temp = 12.5;
A = A + 10;
v = u + f * t;
s = u * t + 0.5 * f * t * t;
```


Contd.

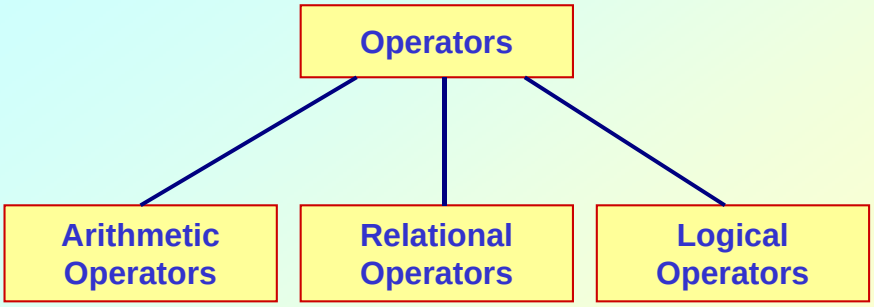
- A value can also be assigned to a variable at the time the variable is declared.

```
int   speed = 30;
char  flag = 'y';
```

- Several variables can be assigned the same value using multiple assignment operators.

```
a = b = c = 5;
flag1 = flag2 = 'y';
speed = flow = 0.0;
```

Operators in Expressions



Arithmetic Operators

- Addition :: +
- Subtraction :: –
- Division :: /
- Multiplication :: *
- Modulus :: %

Examples

```
distance = rate * time ;
netIncome = income - tax ;
speed = distance / time ;
area = PI * radius * radius;
y = a * x * x + b*x + c;
quotient = dividend / divisor;
remain = dividend % divisor;
```


Contd.

- Suppose x and y are two integer variables, whose values are 13 and 5 respectively.

x + y	18
x – y	8
x * y	65
x / y	2
x % y	3

Operator Precedence

- In decreasing order of priority
  1. Parentheses :: ( )
  2. Unary minus :: –5
  3. Multiplication, Division, and Modulus
  4. Addition and Subtraction
- For operators of the *same priority*, evaluation is from *left to right* as they appear.
- Parenthesis may be used to change the precedence of operator evaluation.

Examples: Arithmetic expressions

a + b * c – d / e	➔ a + (b * c) – (d / e)
a * – b + d % e – f	➔ a * (– b) + (d % e) – f
a – b + c + d	➔ (((a – b) + c) + d)
x * y * z	➔ ((x * y) * z)
a + b + c * d * e	➔ (a + b) + ((c * d) * e)

Integer Arithmetic

- When the operands in an arithmetic expression are integers, the expression is called *integer expression*, and the operation is called *integer arithmetic*.
- Integer arithmetic always yields integer values.
- Examples:

```
(12 + 3) / 6          gives the value 2
(2 / 3) * 3           gives the value 0
(12 * 3) / 7 + 3 * 2  gives the value 11
```


Real Arithmetic

- Arithmetic operations involving only real or floating-point operands.
- Since floating-point values are rounded to the number of significant digits permissible, the final value is an approximation of the final result.

  1.0 / 3.0 * 3.0 will have the value 0.99999 and not 1.0
- The modulus operator cannot be used with real operands.

Mixed-mode Arithmetic

- When one of the operands is integer and the other is real, the expression is called a ***mixed-mode*** arithmetic expression.
- If either operand is of the real type, then only real arithmetic is performed, and the result is a real number.

```
25 / 10     gives the value 2
25 / 10.0   gives the value 2.5
```

- Some more issues will be considered later.

Type Casting

- Temporarily convert the type of a variable before being used in an expression.
  - Expressed by specifying the ***desired*** type in parenthesis before the variable/expression.
- Examples:

```
int  a = 10, b = 4, c;   float  x, y;
x = (float) a / b;     /* x will be 2.5 */
y = (float) (a / b);   /* y will be 2.0 */
c = (int) x * 4;       /* c will be 8 */
a = (int) (x * 4);     /* a will be 10 */
```

Relational Operators

- Used to compare two quantities.

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

- The result of comparison is “true” or “false”.
  - The value 0 is considered as “false”, and any non-zero value as “true”.

Examples

```
10 > 20         is false
25 < 35.5       is true
12 > (7 + 5)    is false
```

- When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared.

```
a + b > c – d   is the same as   (a+b) > (c+d)
```

Examples

- Sample code segment in C:

```
if  (x > y)
   printf (“%d is larger\n”, x);
else
  printf (“%d is larger\n”, y);


if (1)   /* will be always true */
   ……
```


Logical Operators

- There are two logical operators in C (also called logical connectives).
  - && ➔ Logical AND
  - || ➔ Logical OR
- What they do?
  - They act upon operands that are themselves logical expressions.
  - The individual logical expressions get combined into more complex conditions that are *true* or *false*.

– **Logical AND**
  - Result is true if both the operands are true.

– **Logical OR**
  - Result is true if at least one of the operands are true.

X	Y	X && Y	X || Y
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

- **Examples:**

```
if  ((i > 2) && (i < 10))
  printf (“\n i lies between 3 and 9”);

if  ((flag == ’A’) || (flag == ’a’)
  printf (“\n Either lower or uppercase A”);
```

Input / Output

- **printf**
  - Performs output to the standard output device (typically defined to be the screen).
  - It requires a format string in which we can specify:
    - The text to be printed out.
    - Specifications on how to print the values.

      printf ("The number is %d.\n", num) ;
    - The format specification %d causes the value listed after the format string to be embedded in the output as a decimal number in place of %d.
    - Output will appear as: The number is 125.

- **scanf**
  - Performs input from the standard input device, which is the keyboard by default.
  - It requires a format string and a list of variables into which the value received from the input device will be stored.
  - It is required to put an ampersand (&) before the names of the variables.

```
scanf ("%d", &size) ;
scanf ("%c", &nextchar) ;
scanf ("%f", &length) ;
scanf (“%d  %d”, &a, &b);
```