C Language Tutorial

(C Language Notes)

Elements of C Language

Yousaf Bux

Topics

- Identifier and Types of Identifiers
- Keywords
- Variable
- Rules for Naming Variable
- Variable Declaration
- Variable Initialization
- Constant
- Data Types (Integer, Float & Character)
- Overflow and Underflow
- Problems with floating point numbers
- Exponential Notation

Identifier

An identifier is the name given to the variable, constant, function or label in the program.

- Feature present in all computer languages
- A good identifier name should be descriptive but short
- An identifier in C language may consist of 31 characters

Rules for Identifier

- The first character must be an alphabet or underscore (_) such as _large, age, marks etc.
- The remaining can be alphabetic characters, digits or underscores such as Number1, total_marks etc.
- The reserved word cannot be used as identifier name such as int, double, break etc.

Valid identifiers letter1, inches, KM_PER_MILE, _large

Invalid identifiers 1letter, Happy*trout, return

Types of Identifiers

1. Standard Identifiers

A type of identifier that has special meaning in C is known as standard identifier.

Examples

printf scanf

2. User-defined Identifiers

A type of identifier that is defined by the programmer is known as user-defined identifier.

- The user-defined identifiers are used to store data and program results

Examples

my_name marks age

Keywords

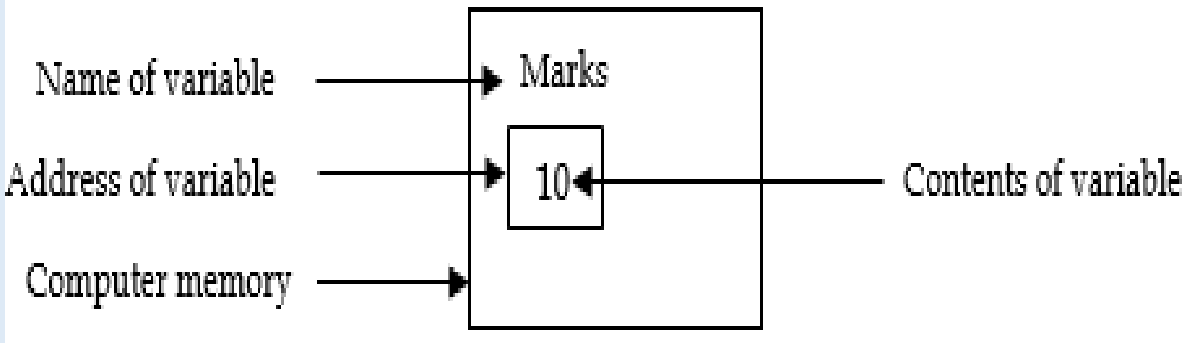
Keyword is a word in C language that has a predefined meaning and purpose.

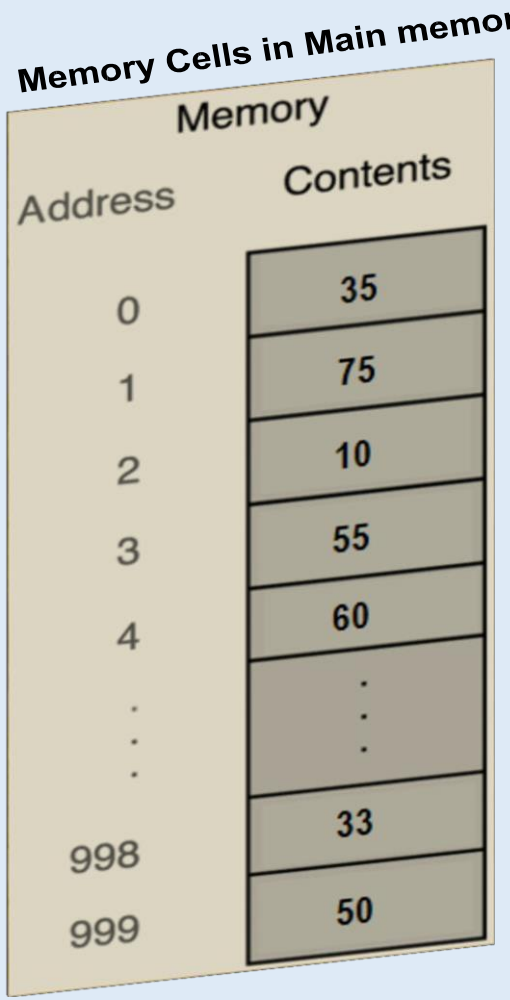
- Also known as reserved words
- Can not be used for another purpose
- Written in lowercase letters
- The total number of keywords is 32

List of Keywords

Variable

- A variable is a named memory location or memory cell
- Store program's input data and its computational results during execution
- The value of variable can change during program execution
- If a new value is stored in the variable, it replaces the previous value
- The variables are created in RAM





Name of variable It refers to an identifier that represents a memory location

Address of variable It refers to the memory location of the variable

Contents of variable It refers to the value stored in memory location referred by variable

Rules for Naming Variables

Rules	Examples
• Variable may include letters, numbers and underscore (_)	sub1, f_name, Number1
• First character of variable must be a letter or underscore _	_area , age
• Blank spaces are not allowed in variable names	My height , total marks
• Special symbols such as @, $, % are not allowed in variable names	$cost , tot.al , success!
• Variables are case sensitive	Tax != tax
• Reserved word cannot be used as variable name	long, double, void
• A variable can be declared only for one data type	
• A variable can be up to 31 characters long for many compilers	

Multi-word Variable Names

- A variable name should reflect its purpose
  - A variable to store the marks of student should be **Marks.**
- Descriptive variable names may include multiple words
- Two conventions to use in naming variables:
  - Capitalize the first letter of each word.

    InterestRate **TotalSales**
  - Use the underscore _ character as a space:

    My_Phone_No **Total_Sales**
- Use one convention consistently throughout a program

Valid and Invalid Variable

- income — **Valid.**
- double — **Invalid.** A keyword cannot be used as variable name.
- total marks — **Invalid.** A variable cannot have spaces in it.
- averge-score — **Invalid.** A variable cannot have hyphen in it.
- 2ndTry — **Invalid.** A variable cannot start with a digit.
- $cost — **Invalid.** The special symbol $ cannot be used in variable name.
- MAX_SPEED — **Valid**
- My.school — **Invalid.** A variable cannot have a period "." in it.
- room# — **Invalid.** A variable cannot have hash sign in it.
- no_of_students — **Valid.**

Variable Declaration

The process of specifying the variable name and its type is called variable declaration.

- All variables must be declared before they are used in the program
- A program can have as many variables as needed
- The variable declaration provides information to the compiler about the vari

Remember!
The compiler gives an error if an undeclared variable is used in the program

- The compiler uses the declaration to determine how much memory is needed for each variable
  - **Example** – int variable requires 2 bytes and char variable requires 1 byte.

Once a variable is declared, its data type cannot be changed during program execution

Remember!

The value of the variable can be changed during program execution

Syntax

```
data_type variable_name;
```

data_type	It indicates type of data that can be stored in variable
variable_name	It refers to the memory location of the variable

Variable Declaration (cont.)

- Variables of the same type can be declared:
  - In separate statements

    int length;

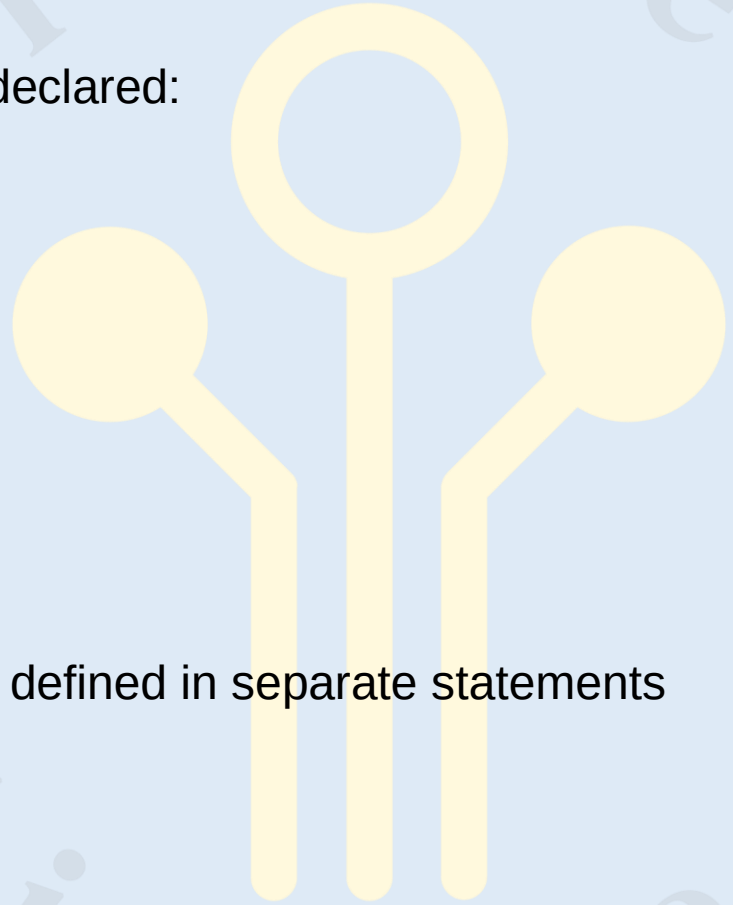
    int width;
  - In the same statement

    int length, width;
- Variables of different types must be defined in separate statements

  int marks;

  float average;

  char grade;



Variable Initialization

The process of assigning value to a variable at the time of declaration is called variable initialization.

- Symbol = is used to initialize a variable
- A single variable name must appear on left side of the = symbol
- **Garbage value**
  - If a variable declared but not initialized, it may contain meaningless data
  - Produce unexpected results in some computations
  - All variables should be initialized to avoid this problem

Syntax

type_name variable = value;

Examples

int n = 100;

int x = 50, y = 75;

float average = 50.73;

char grade = ‘A’;

Constant

A constant is a quantity that cannot be changed during program execution.

Types of Constants

1. Numeric Constants

Numeric constant consists of numbers. It can be further divided into two types:

a) Integer Constant

Integer constants are numeric values without fraction or decimal point.

Examples

87	45	-10	-5

b) Floating Point Constants

Floating point constants are numeric values with fraction or decimal point.

Examples

50.75	10.22	–13.4

Constants (cont.)

2. Character Constants

Any character written within single quotation mark is known as character constant.

Examples

‘A’ ‘n’ ‘9’ ‘=’ ‘$’

3. String Constants

A collection of characters written in double quotations mark is called string or string constant.

- May consist of any alphabetic characters, digits and special symbols

Examples

“Pakistan” “123” “99-Mall Road, Lahore”

Data Types

The data type specifies the type of data and set of operations that can be applied on the data.

- Every data type has a range of values
- Requires different amount of memory
- Compiler allocates memory space for each variable according to its data type

1. Standard Data Type

A data type that is predefined in the language is called standard data type.

Examples

int float long double char

2. User-defined Data Type

C also allows the user to define his own data types known as user-defined data type.

- Can only be used in that program in which it has been defined

Data Types

The data type specifies the type of data and set of operations that can be applied on the data.

- Every data type has a range of values
- Requires different amount of memory
- Compiler allocates memory space for each variable according to its data type

Categories of Data Types

1. Standard Data Type

A data type that is predefined in the language is called standard data type.

Examples

int float long double char

2. User-defined Data Type

C also allows the user to define his own data types known as user-defined data type

- Can only be used in that program in which it has been defined

Integer Data Type

Integer data is the numeric value with no decimal point or fraction.

- May have positive or negative value

Examples

10 520 -20

Types of Integer Data Type

int data type is used to represent integer values

short data type is used to store integer values

unsigned int data type is used to store only positive integer values

long data type is used to store large integer values

unsigned long data type is used to store large positive integer values

Data Type	Size in Bytes	Description
int	2	Ranges from -32,768 to 32,767.
short	2	Ranges from -32,768 to 32,767.
unsigned int	2	Ranges from 0 to 65,535.
long	4	Ranges from -2,147,483,648 to 2,147,483,647.
unsigned long	4	Ranges from 0 to 4,294,967,295.

Floating Point Data Type

The floating point data is a numeric value with decimal point or fraction.

- Consists of integer and non-integer part
- Also called real number

Examples

10.5 5.3 -10.91

precision refers the number of digits after the decimal point

Types of Floating point Data Type

float data type is used to store real values

- Provides the precision of 6 decimal places

double data type is used to store large real values

- Provides the precision of 15 decimal places

long double data type is used to store very large real values

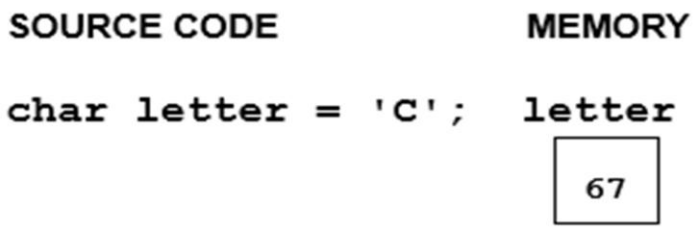
- Provides the precision of 19 decimal places

Data Type	Size in Bytes	Description
float	4	$3.4x10^{-38}$ to $3.4x10^{+38}$
double	8	$1.7x10^{-308}$ to $1.7x10^{+308}$
long double	10	$1.7x10^{-4932}$ to $1.7x10^{+4932}$

Character Data Type

char data type is used to store character value.

- Takes 1 byte in memory
- Used to represent:
  - Letter a - z and A - Z
  - Numbers 0-9
  - Space (blank)
  - Special characters , . ; ? “ / ( ) [ ] { } * & % ^ < > etc.
- Character values are normally given in single quotes E.g. ‘A’, ‘*’
- Numeric code (ASCII) representing the character is stored in memory



- Possible to perform mathematical operation on character values

Example

```
#include <stdio.h>
#include <conio.h>
void main()
{
  char ch1, ch2, sum;
  clrscr();
  ch1 = '2';
  ch2 = '6';
  sum = ch1 + ch2;
  printf("Sum = %d", sum);
  getch();
}
```

Output:
Sum = 104

ASCII values of '2' and '6' are 50 and 54

Using signed & unsigned Keywords with char Data Type

- The keywords ***signed*** and ***unsigned*** can also be used with ‘char’ type data.
- By default, the ‘char’ data type is unsigned.
- ASCII values of alphabets, numbers and other special characters are positive.
- The value range for unsigned ‘char’ type data is from 0 to 255.
- The value range for signed char type data is from –128 to 127.

Data Types	Memory Size	Range
char (by default signed)		
signed char	1 byte	−128 to 127
unsigned char	1 byte	0 to 255
Short (by default signed)		
signed short	2 byte	−32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int (by default signed)		
signed int	2 byte	−32,768 to 32,767
unsigned int	2 byte	0 to 65,535
short (by default signed)		
signed short int	2 byte	−32,768 to 32,767
unsigned short int	2 byte	0 to 65,535

long (by default signed)		
signed long	4 byte	-9223372036854775808 to 9223372036854775807
unsigned long	4 byte	0 to 18446744073709551615
float (6 decimal places)	4 byte	$3.4 * 10^{-38}$ to $3.4 * 10^{+38}$
double (15 decimal places)	8 bytes	$1.7 * 10^{-308}$ to $1.7 * 10^{+308}$
long double (19 decimal places)	10 bytes	$1.7 * 10^{-4932}$ to $1.7 * 10^{+4932}$

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <float.h>

int main(int argc, char** argv) {

   printf("CHAR_BIT   :   %d\n", CHAR_BIT);     printf("CHAR_MAX   :   %d\n", CHAR_MAX);     printf("CHAR_MIN   :   %d\n", CHAR_MIN);
   printf("INT_MAX    :   %d\n", INT_MAX);    printf("INT_MIN    :   %d\n", INT_MIN);    printf("LONG_MAX   :   %ld\n", (long) LONG_MAX);
   printf("LONG_MIN   :   %ld\n", (long) LONG_MIN);    printf("SCHAR_MAX  :   %d\n", SCHAR_MAX);    printf("SCHAR_MIN  :   %d\n", SCHAR_MIN);
   printf("SHRT_MAX   :   %d\n", SHRT_MAX);    printf("SHRT_MIN   :   %d\n", SHRT_MIN);
   printf("UCHAR_MAX  :   %d\n", UCHAR_MAX);
   printf("UINT_MAX   :   %u\n", (unsigned int) UINT_MAX);
   printf("ULONG_MAX  :   %lu\n", (unsigned long) ULONG_MAX);
   printf("USHRT_MAX  :   %d\n", (unsigned short) USHRT_MAX);

   return 0;
}
```

Overflow and Underflow

- **Overflow** occurs when value assigned to the variable is more than the maximum possible value
  - **Example -** The maximum value for **int** type variable is 32767. If the assigned value is more than 32767, then an integer *overflow* occurs
- An **underflow** occurs when the value assigned to a variable is less than the minimum possible value
  - **Example -** The minimum value of **int** type variable is -32768.
    If the assigned value is less then -32768 then an integer underflow occurs.
- Overflow and underflow is not detected by the compiler
- Program may produce wrong result.
- Using variables with appropriately-sized data types can minimize this problem

Program 9.1

Write a program that explains the concept of overflow and underflow.

```
#include <conio.h>
#include <stdio.h>
void main()
{
   short testVar = 32767;
   clrscr();
   printf("%d \n", testVar);
   testVar = testVar + 1;
   printf("%d \n", testVar);
   testVar = testVar - 1;
   printf("%d \n", testVar);
   getch();
}
```

Output:

```
32767
-32768
32767
```

Problems When Manipulating Floating Point Numbers

- **Cancellation error** is an error resulting from applying an operation to a large number and a small number, and the effect of the smaller number is lost.

  e.g., **1000.0 + 0.0000001234 is equal to 1000.0**

  197.0 + 0.000000134 is equal to 1947.0

- **Arithmetic Underflow** is an error in which a very small computational result is represented as zero

  e.g., **$0.00000001 * 10^{-1000000}$ is equal to 0**

- **Arithmetic Overflow** is an error in which the result is too large to be represented
  - Some machine may let the arithmetic overflow occur and the computational result is unexpected

  e.g., **$999999999 * 10^{9999999}$** may become a negative value in some machine

Exponential Notation

- The exponential notation consists of two parts:
  - Mantissa
  - Exponent
- General form of writing floating point values in exponential notation is:

  ±m e ±n OR ±m E ±n
- The value before e is known as **mantissa**. It has absolute value greater than or equal to 1 and less than 10
- The value after e is known as exponent and always have integer value
- Mantissa and exponent may be positive or negative value

Examples

Floating Point Number	Scientific Notation	Exponential Notation
1.98	1.98×10^{0}	1.98e0
4,679,000.0	4.679×10^{6}	4.679e6
0.00053	5.3×10^{-4}	5.3e-4

Range and Precision

- The possible values that a floating type variable can store are described in terms of precision and range:

Precision	It is the number of digits after the decimal point
Range	It is the exponential power of 10

- **float** variable has a precision of 6 digits and a range of 10^{-38} to 10^{+38}
- **double** variable has a precision of 15 digits and a range of 10^{-308} to 10^{+308}
- A large precision provides more accuracy

Operators

Operators are the symbols that are used to perform different operations on data.

Types of Operators

- Arithmetic Operators — **+, -, * , / , and %**
- Relational Operators — **>, <, ==, >=, <=, and !=**
- Logical Operators — **&&, || and !**
- Assignment Operators — **=**
- Increment and Decrement Operators — **++, - -**
- Compound Assignment Operators — **+=, -=, *= , /= and %=**

Expression

A statement that evaluates to a value is called an expression.

- Gives a single value
- Consists of operator and operand
  - An operator is a symbol that performs some operation
  - Operand is the value on which the operator performs some operation
- Can be a constant, variable or combination of both

Examples

A + B;

m / n;

x + 100;

+ is the operator

A and **B** are variables used as operands

Unary and Binary operator

Unary Operators

A type of operator that works with one operand is called unary operator.

- Some unary operators are –, ++, – –, !
- The above operators are used with one operand

```
a;
N++;
– –x;
```

Binary Operators

A type of operator that works with two operands is called binary operator.

- Some binary operators are +, –, *, /, %, >, && (AND), || (OR)
- The above operators are used with two operands

```
a + b;
x / y;
a > b;
```

Arithmetic Operator

Arithmetic operator is a symbol that performs mathematical operation on data.

Operation	Symbol	Description
Addition	+	Adds two values
Subtraction	–	Subtracts one value from another value
Multiplication	*	Multiplies two values
Division	/	Divides one value by another value
Modulus	%	Gives the remainder of division of two integers

% Operator

- Also called **remainder operator**
- modulus operator (%) works only with integer values
- Computes the remainder resulting from integer division

  7 % 5 → 2 **12 % 3 → 0**

- If modulus operator is used with the division of 0, the result will always be 0
- In expression like 3 % 5, 3 is not divisible by 5. Its result is 3

/ Operator

- C division operator (`/`) performs integer division if both operands are integers
- fractional part of the quotient is truncated

Examples

The result of 7 / 4 evaluates to 1 and 17 / 5 evaluates to 3

- If either operand is floating-point, the result is floating-point

Examples

The result of 7.0/2.0 is 3.5 and 13 / 5.0 is 2.6

5.0 / 2 → 2.5 4.0 / 2.0 → 2.0 17.0 / 5.0 → 3.4

Arithmetic Expression

A type of expression that consists of constants, variables and arithmetic operators is called arithmetic expression.

Examples

Suppose we have two variables A and B where A = 10 and B = 5.

Arithmetic Expression	Result
A + B	15
A – B	5
A * B	50
A / B	2
A % B	0

Data Type of Expression

- The data type of each variable must be specified in its declaration, but how does C determine the data type of an expression?
- What is the type of expression x+y when both x and y are of type int?
  - The data type of an expression depends on the type(s) of its operands
  - The result of ex

    pression is evaluated to larger data type in the expression
  - If both are of type int, then the expression is of type int.
  - If either one or both is of type double,

    then the expression is of type double.

Expression	Data Type of Expression
int + float	float
int – long	long
int * double	double
float / long double	long double

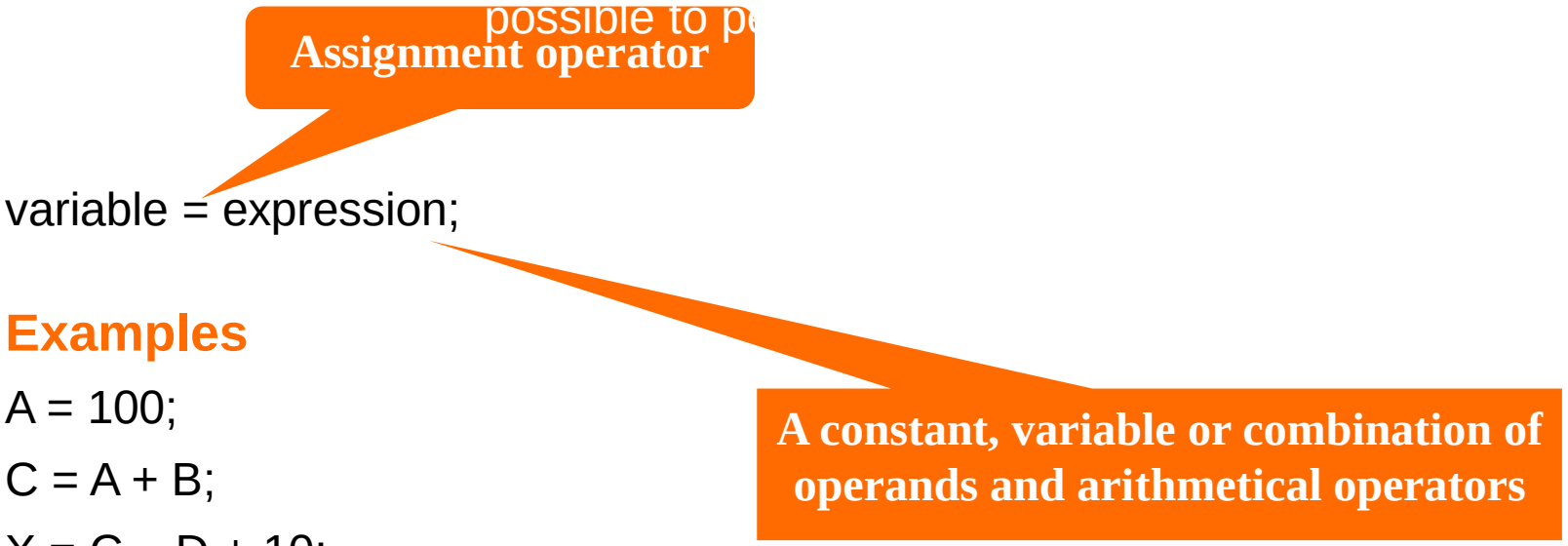
- An expression in which operands are of different data types is called **mixed-type expression**.
  - An expressions that has operands of both int and double is a mixed-type expression

Assignment Statement

A statement that assigns a value to a variable is known as assignment statement.

- When an assignment statement is executed, the expression is first evaluated; then the result is assigned to the variable to the left side of assignment operator

Syntax



Examples

A = 100;

C = A + B;

X = C – D + 10;

Lvalue and Rvalue

- An **lvalue** is an operand that can be written on the left side of assignment operator **=**
- An **rvalue** is an operand that can be written on the right side of assignment operator **=**
- Lvalues must be variables
- rvalues can be any expression

Example

- distance = rate * time;
  lvalue: "distance"
  rvalue: "rate * time"
- All lvalues can be used as rvalues but all rvalues cannot be used as lvalues
  - **x = 5** is valid but **5 = x** is not valid.
  - C = A + B is valid but A+B=C is invalid.

Compound Assignment Statement

An assignment statement that assigns a value to many variables is known as compound assignment statement.

- Assignment operator = is used in this statement

Example

```
A = B = 10;
```

- assigns the value 10 to both A and B.

- Compound assignment operators combine assignment operator with arithmetic operators
- Used to perform mathematical operations more easily.

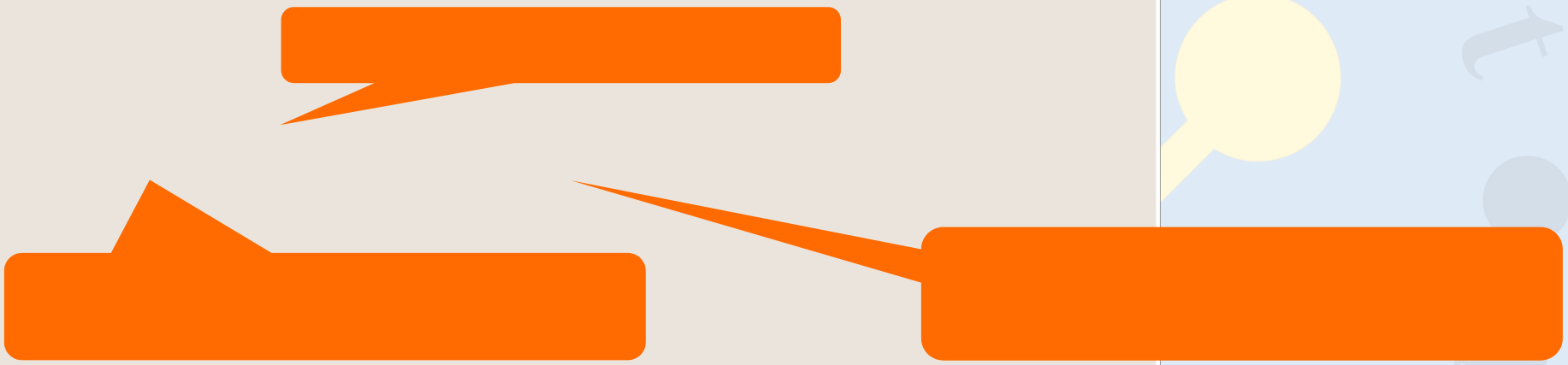
Syntax

variable op = expression;

Example

Operator	Example	Equivalent to
+=	A += 10	A = A + 10
-=	A -= 10	A = A - 10
*=	A *= 10	A = A * 10
/=	A /= 10	A = A / 10
%=	A %= 10	A = A % 10

Compound Assignment Operator



Program 9.3

Write a program that performs all compound assignment operations on an integer variable.

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a;
  clrscr();
  a = 10;
  printf("Value of a: %d \n", a);
  a += 5;
  printf("Value of a after a+=5: %d \n", a);
  a -= 5;
  printf("Value of a after a-=5: %d \n", a);
  a *= 2;
  printf("Value of a after a*=2: %d \n", a);
  a /= 2;
  printf("Value of a after a/=2: %d \n", a);
  a %= 2;
  printf("Value of a after a%%=2: %d \n", a);
  getch();
}
```

Output:

```
Value of a : 10
Value of a after a+=5 : 15
Value of a after a-=5 : 10
Value of a after a*=2 : 20
Value of a after a/=2 : 10
Value of a after a%=2 : 0
```

Increment Operator

- Increment operator, ++ is a unary operator and works with single variable.
- Used to increase the value of variable by 1 and is denoted by the symbol ++

  A++; is equivalent to A = A + 1;
- Increment operator cannot increment the value of constant and expressions

  Example 10++ , (a+b)++ or ++ (a+b) are invalid

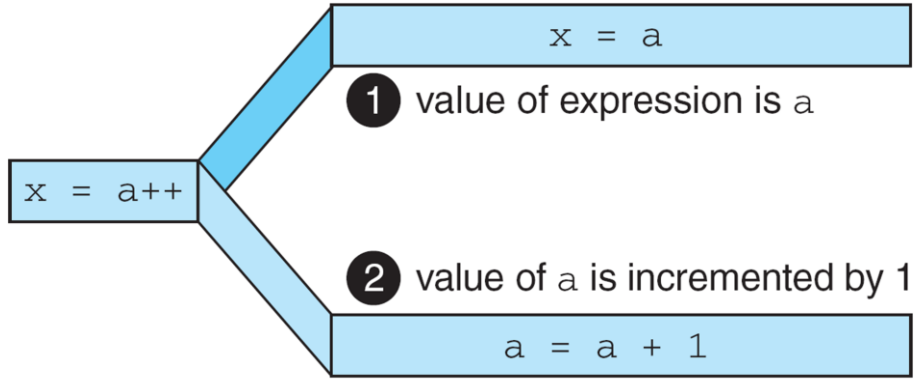
Prefix and Postfix Increments

- No difference if 'alone' in statement:
  - A++; and ++A; → identical result
- The value of the expression (that uses the ++ operators) depends on the position of the operator
  - **Example -** Result of two statements x=a++ and x=++a are different

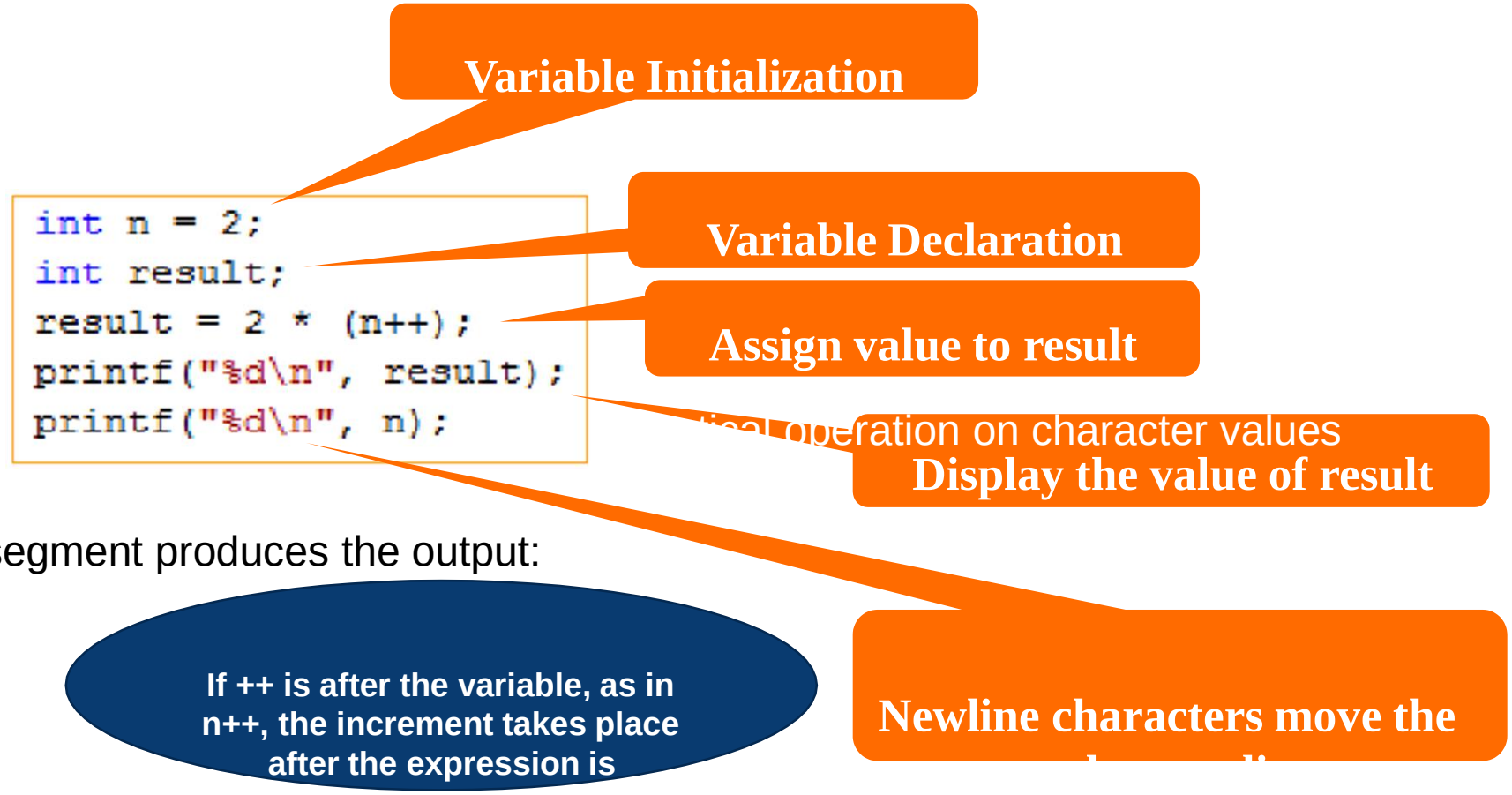
Postfix increment in Expressions

- Uses or assign current value of variable, THEN increment

If ++ is after the variable, as in a++, the increment takes place after the expression is evaluated



Example:



- This code segment produces the output:

4

3

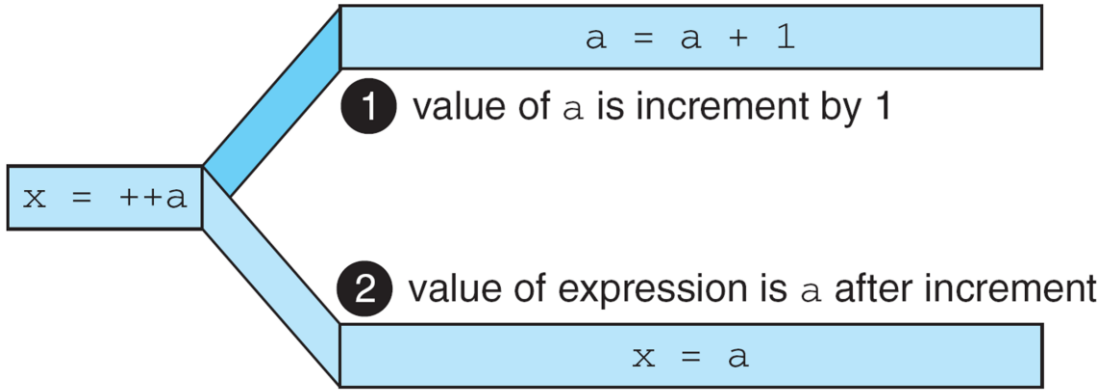
If ++ is after the variable, as in n++, the increment takes place after the expression is

- Since postfix increment was used

Prefix increment in Expressions

- Increments variable first, THEN uses new value

If ++ is before the variable, as in ++a, the increment takes place before the expression is evaluated



Example:

```
int n = 2;
int result;
result = 2 * (++n);
printf("%d\n", result);
printf("%d\n", n);
```

- This code segment produces the output:

  6

  3

- Since pre-increment was used

Program 9.4

Write a program that explains the difference of postfix increment operator and prefix increment operator used as independent expression.

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int a, b, x, y;
   clrscr();
   a = b = x = y = 0;
   a++;
   b = a;
   ++x;
   y = x;
   printf("a = %d \n b = %d \n", a, b);
   printf("x = %d \n y = %d \n", x, y);
   getch();
}
```

Output:

```
a = 1
b = 1
x = 1
y = 1
```

Program 9.5

Write a program that explains the difference of postfix increment operator and prefix increment operator used as part of a larger expression.

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int a, b, x, y;
   clrscr();
   a = b = x = y = 0;
   b = a++;
   y = ++x;
   printf("a = %d \n b = %d \n", a, b);
   printf("x = %d \n y = %d \n", x, y);
   getch();
}
```

Output:

```
a = 1
b = 0
x = 1
y = 1
```

Decrement Operator

- Decrement operator, is a unary operator and works with single variable
- Used to decrease the value of variable by 1 and is denoted by the symbol - -

  A--; is equivalent to A = A – 1;
- Decrement operator cannot decrement the value of constant and expressions.

  Example - -10 , (a+b) - - or - - (a+b) are invalid

Prefix and Postfix Decrements

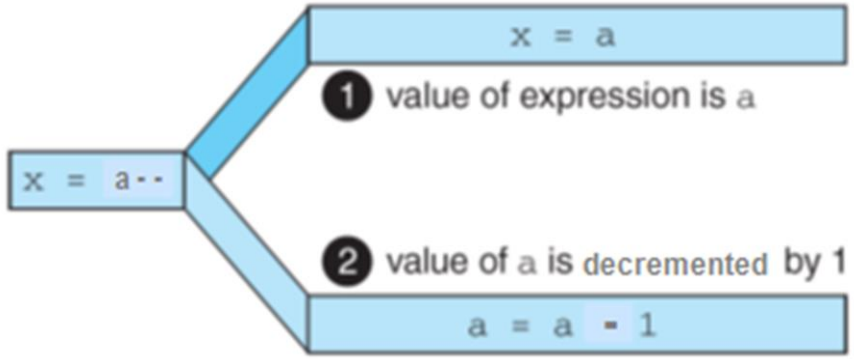
- No difference if 'alone' in statement: A--; and --A; -> identical result
- The value of the expression (that uses the -- operators) depends on the position of the operator.

  Example Result of two statements x = a-- and x = --a are different.

Postfix decrement in Expressions

- Uses current value of variable, THEN decrements it

If - - is after the variable, as in a- -, the increment takes place after the expression is evaluated



Example:

```
int n = 2 ;
int result;
result = 2 * (n--) ;
printf ( "%d\n", result);
printf("%d\n", n );
```

Variable Initialization

Variable Declaration

Assign value to result

Display the value of result

Newline characters move the cursor to the next line

- This code segment produces the output:

4

1

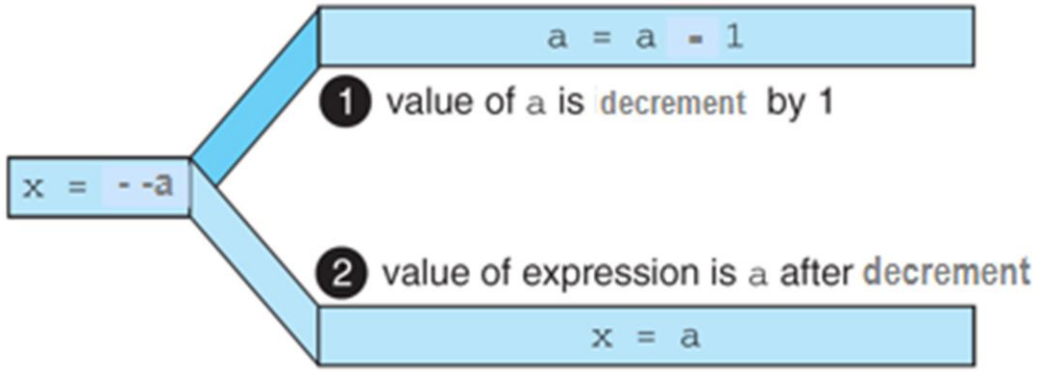
If -- is after the variable, as in n--, the decrement takes place after the expression is evaluated

- Since postfix decrement was used

Prefix decrement in Expressions

- Decrements variable first, THEN uses new value

If -- is before the variable, as in --a, the decrement takes place before the expression is evaluated



Example:

```
int n = 2 ;

int result;

result = 2 * (--n) ;

printf ( "%d\n", result);

printf("%d\n", n );
```

- This code segment produces the output:

  2
  1

- Since pre-decrement was used

Program 9.6

Write a program that explains the difference of postfix decrement operator and prefix decrement operator used as independent expression.

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a, b, x, y;
  clrscr();
  a = b = x = y = 0;
  a--;
  b = a;
  --x;
  y = x;
  printf("a = %d \n b = %d \n", a, b);
  printf("x = %d \n y = %d \n", x, y);
  getch();
}
```

Output:

```
a = - 1
b = -1
x = -1
y = -1
```

Program 9.7

Write a program that explains the difference of postfix decrement operator and prefix decrement operator used as part of a larger expression.

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a, b, x, y;
  clrscr();
  a = b = x = y = 0;
  b = a--;
  y = --x;
  printf("a = %d \n b = %d \n", a, b);
  printf("x = %d \n y = %d \n", x, y);
  getch();
}
```

Output:

```
a = -1
b = 0
x = -1
y = -1
```

Relational Operators

- Used to specify conditions in programs
  - Condition is an expression that is either false (represented by 0) or true (represented by 1).
    e.g., "**grade > 60**" is a condition.
  - Conditions contai **relational operators**, and have the following forms

    variable relational-operator variable a>b

    variable relational-operator constant marks > 75

    possible t perform mathematical operation on character v

- Also called **comparison operators** as they test conditions that are either **true** or **false**

Operators Used in Conditions

Operator	Meaning
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Relational Expression

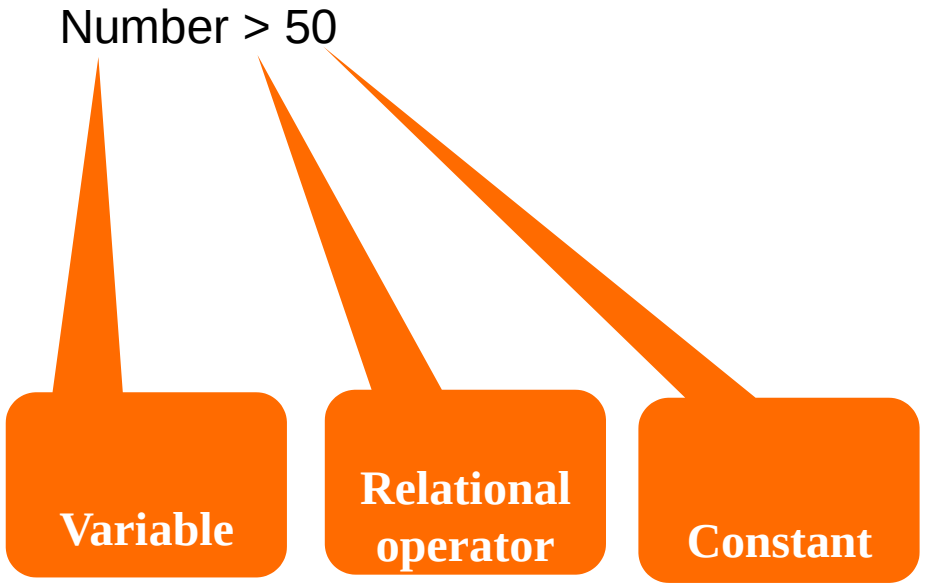
A type of expression that consists of constants, variables and relational operators is called relational expression.

- Expressions are used to compare the values
- The result of a relational expression can be true or false

Examples

Suppose the value of A = 10 and the value of B = 5.

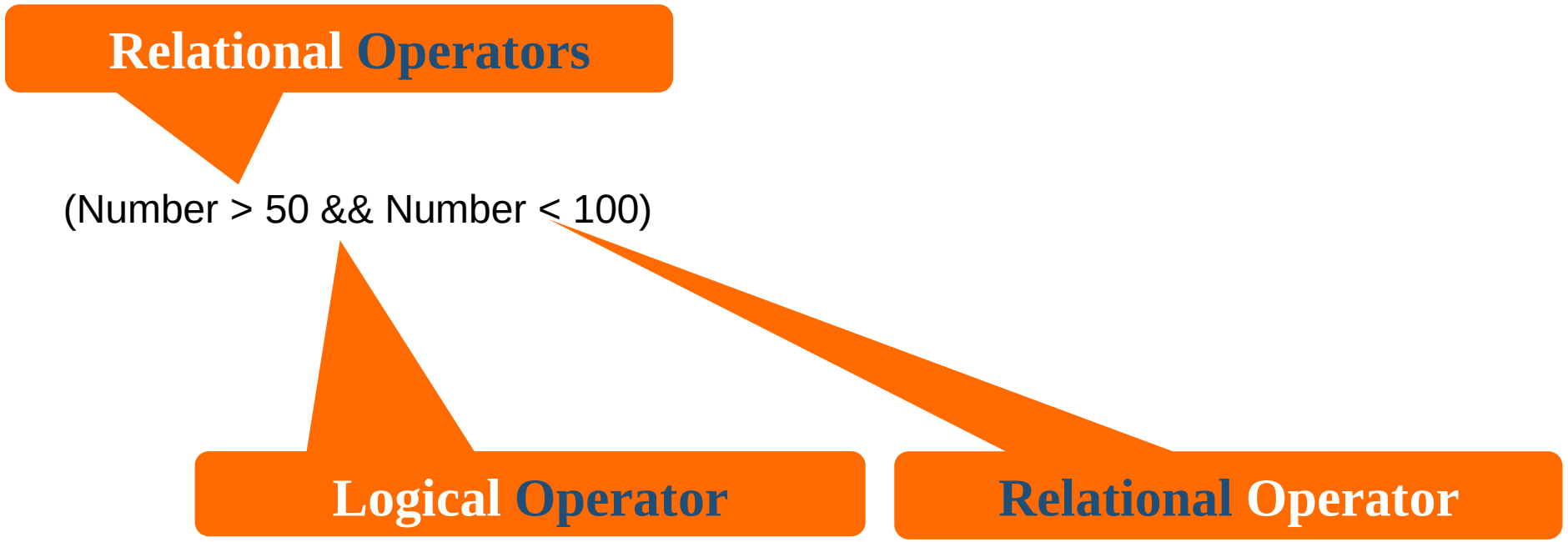
Relation Expression	Result
A > B	True
A < B	False
A <= B	False
A >= B	True
A == B	False
A != B	True



Compound Condition

A type of comparison in which more than one conditions are evaluated is called compound condition

- **Example** Program inputs two numbers to displays OK if one number is greater than 50 and second number is less than 100.
- Compound condition is executed by using logical operators.



Logical Operator

The logical operators are used to evaluate compound conditions.

- Types of logical operators

  && and
  || or
  ! not

AND Operator (&&)

- Used to evaluate two conditions
- Produces **true** result if both conditions are **true**
- produces **false** result if any one condition is **false**

Condition 1	Operator	Condition 2	Result
False	&&	False	False
False	&&	True	False
True	&&	False	False
True	&&	True	True

Example

Two variables A = 100 and B = 50

- Compound condition **(A>10) && (B>10)** is **true**
- Compound condition **(A>50) && (B>50)** is **false**

Logical Operator(Cont.)

AND Operator (| |)

- Used to evaluate two conditions
- Produces **true** result if either conditions is **true**
- Produces **false** result if both conditions are **false**

Condition 1	Operator	Condition 2	Result
False	||	False	False
False	||	True	True
True	||	False	True
True	||	True	True

Example

Two variables A = 100 and B = 50

- Compound condition **(A>10) | | (B>10)** is **true**
- Compound condition **(A>50) | | (B>50)** is **true**
- Compound condition **(A>200) | | (B >100)** is **false**

Logical Operator(Cont.)

NOT Operator (!)

- Reverse the result of a condition
- Produces **true** result if conditions is **false**
- Produces **false** result if condition is **true**

Operator	Condition	Result
!	True	False
!	False	True

Example

Two variables A = 100 and B = 50

- Condition **!(A==B)** is **true.**
- Condition **!(A>B)** is **false.**

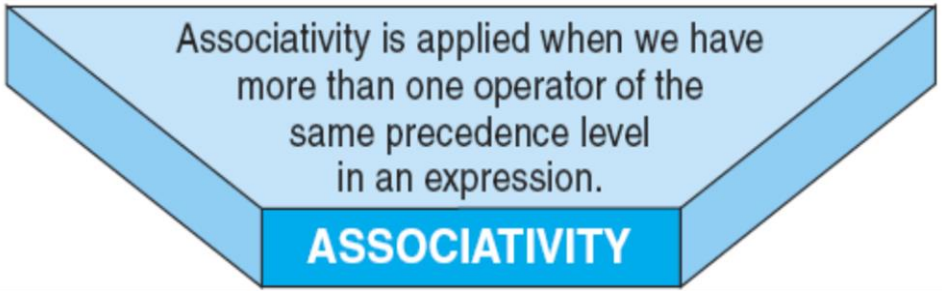
Result of condition (A==B) is false but NOT operator converts it into true

Result of condition (A>B) is true but NOT operator converts it into false

Operator Precedence and Associativity

The order in which different types of operators in an expression are evaluated is known as operator precedence.

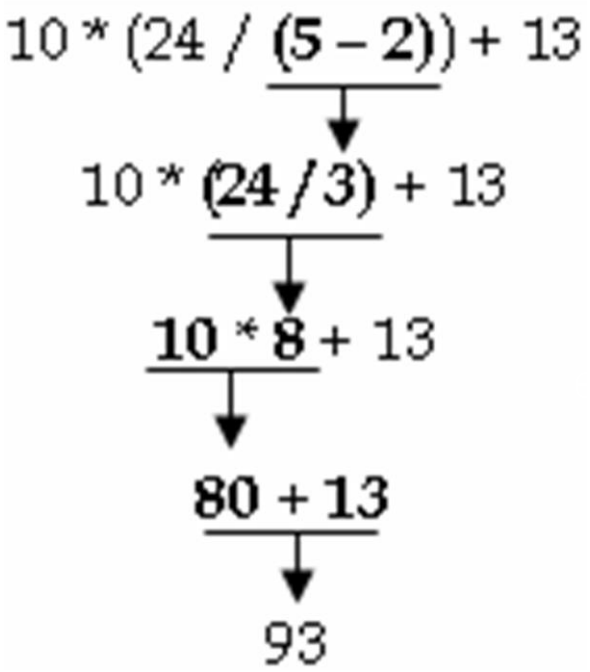
- also known as **hierarchy of operators**
- Any expression given in parentheses is evaluated first
- In case of parentheses within parentheses,
  the expression of the inner parentheses will be evaluated



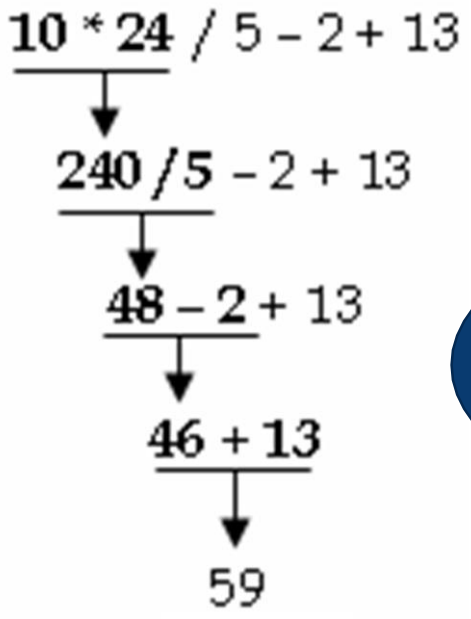
Operator	Operator Name	Direction	Precedence
!	Logical Not	Left to Right	Highest
*, /, %	Multiplication,division, Modulus	Left to Right	
+, -	Addition, subtraction	Left to Right	
<, <=, >, >=	Relational operator	Left to Right	
==, !=	Relational operator	Left to Right	
&&	Logical AND	Left to Right	
||	Logical OR	Left to Right	
+=, -+, *=, /=, %=	Compound assignment	Right to Left	
=	Assignment Operator	Right to Left	Lowest

Example

Operator precedence



Operator associativity



*** and / have equal precedence. These expressions will be evaluated from left to right. Similarly, + and - also have equal precedence.**

aracter values

- First of all, the expression 5–2 will be evaluated. It gives a value 3
- Secondly, 24 will be divided by the result of last line i.e. 24 / 3 giving value 8
- Thirdly, 10 will be multiplied by 8 i.e. giving a result 80
- Finally, 80 will be added in 13 and the last result will be 93.

Comments

- Explanatory lines of a program that help user to understand the source code.
- Make programs easy to read and modify
- Are ignored by the compiler.
- Can be added anywhere in the program.

- Comments can be added in the program in two ways.

    1. Sing-line Comments
    2. Multi-line Comments

Single-line comment

- Begin with // and continue to the end of line.

```
// Program 2.5: Finding the area of rectangle
int length = 12; // length in inches
int width = 15;  // width in inches
int area;        // calculated area
// Calculate rectangle area
area = length * width;
```

Multi-Line Comments

- Begin with /* and end with */
- Can span multiple lines.

```
/* Program 2.7
   Description: Finding the area of circle

   Written by: Abdullah */
```

- Can also be used as single-line comments.

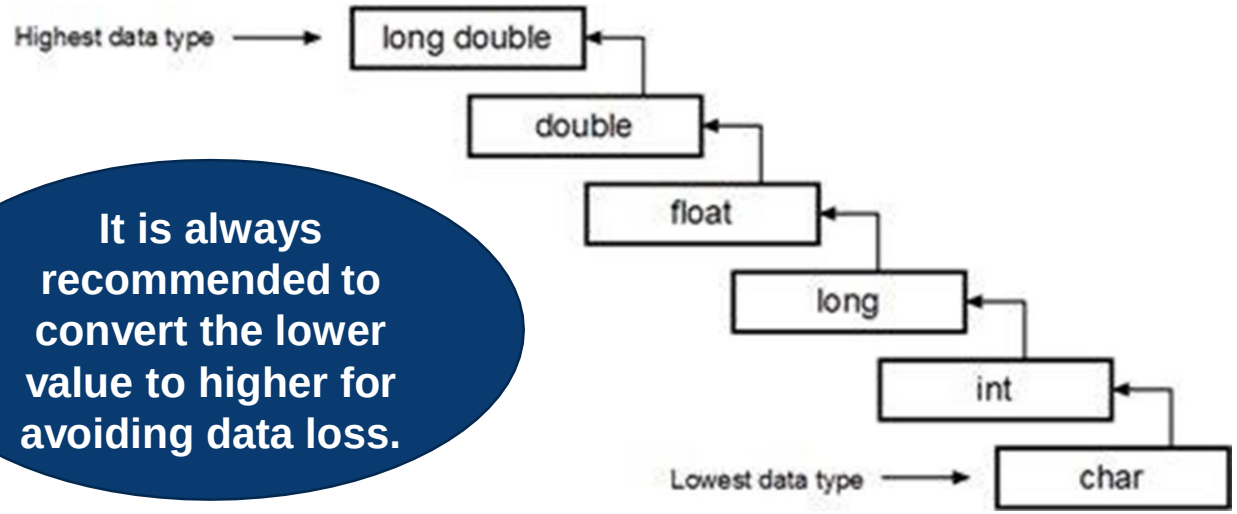
```
int area;   /* Calculated area */
```

Type Casting

- Way of changing a value of one type to a value of another type.
- Two types of casting:

1. Implicit Type Casting

- Performed automatically by the C++ compiler
- when value of one type is automatically changed to another type
- Operations are performed between operands of the same type.
- If the data types of operands are different, the value with lower data type is converted into higher data type.



It is always recommended to convert the lower value to higher for avoiding data loss.

Example

```
int x=20;
char y='a'
// y implicitly converted to int. ASCII
   // value of 'a' is 97
   x = x + y;
printf("x= %d" , x);
```

2. Explicit Type Casting

- Explicit casting is performed by programmer
- C allows the programmer to convert the type of an expression
- It is performed by using cast operator

Syntax

(type) expression;

- Placing the desired type in parentheses before the expression
- When casting from double to int,
  the decimal portion is just truncated – not rounded.

Example

```
double x=3.6;
int sum;
// Explicit conversion from double to int
sum = (int) x + 1;
printf("sum = %d", sum);
```

Program 9.8

/*Write a program that divides two float variables and finds the remainder by using explicit casting. */

```
#include <stdio.h>
#include <conio.h>
void main()
{
  float a, b;
  int c;
  clrscr();
  a = 10.3;
  b = 5.2;
  c = (int)a % (int)b;
  printf("Result is %d", c);
  getch();
}
```

Output:
Result is 0