


Electrical and Electronic Programming

Professor Mannan Saeed Muhammad
College of Information and Communication Engineering

Input /Output Operations and Functions

- input operation
  - an instruction that copies data from an input device into memory
- output operation
  - an instruction that displays information stored in memory
- input/output function
  - a C function that performs an input or output operation
- function call
  - calling or activating function

Input / Output Functions

- A C function that performs an input or output operation
- A few functions that are pre-defined in the header file stdio.h such as :
  - printf()
  - scanf() → MSVS2017 ⇒ scanf_s()
  - getchar() & putchar()

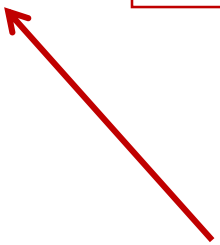
The printf Function

- Used to send data to the standard output (usually the monitor) to be printed according to specific format.
- General format:
  - printf(“string literal”);
    - A sequence of any number of characters surrounded by double quotation marks.
  - printf(“format string”, variables);
    - Format string is a combination of text, conversion specifier and escape sequence.

The printf Function

- function argument
  - enclosed in parentheses following the function name
  - provides information needed by the function

```
printf("That equals %f kilometers. \n", kms);
```



function name

The printf Function

- Example:
  - printf(“Thank you”);
  - printf (“Total sum is: %d\n”, sum);
    - %d is a placeholder (conversion specifier)
      - marks the display position for a type integer variable
    - \n is an escape sequence
      - moves the cursor to the new line

The printf Function

- format string
  - in a call to printf, a string of characters enclosed in quotes, which specifies the form of the output line

```
printf("That equals %f kilometers. \n", kms);
```

The printf Function

- print list
  - in a call to printf, the variables or expressions whose values are displayed
- placeholder/conversion specifier
  - a symbol beginning with % in a format string that indicates where to display the output value

```
printf("That equals %f kilometers. \n", kms);
```

Escape sequence

Placeholder / Conversion Specifier

Conversion Specifier	Variable Type	Function Use
% c	char	printf/scanf_s
% d	int	printf/scanf_s
% f	double	printf
% lf	double	scanf_s

Placeholder / Conversion Specifier

No	Conversion Specifier	Output Type	Output Example
1	%d	Signed decimal integer	76
2	%i	Signed decimal integer	76
3	%o	Unsigned octal integer	134
4	%u	Unsigned decimal integer	76
5	%x	Unsigned hexadecimal (small letter)	9c
6	%X	Unsigned hexadecimal (capital letter)	9C
7	%f	Integer including decimal point	76.0000
8	%e	Signed floating point (using e notation)	7.6000e+01
9	%E	Signed floating point (using E notation)	7.6000E+01
10	%g	The shorter between %f and %e	76
11	%G	The shorter between %f and %E	76
12	%c	Character	‘7’
13	%s	String	‘76'

Escape Sequence

Escape Sequence	Effect
\a	Beep sound
\b	Backspace
\f	Formfeed (for printing)
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\\	Backslash
\”	“ sign
\o	Octal decimal
\x	Hexadecimal
\O	NULL

The scanf_s function

- Read data from the standard input device (usually keyboard) and store it in a variable.
- General format:
  - scanf_s("Format string", &variable);
- Notice ampersand "&" operator :
  - C address of operator
  - it passes the address of the variable instead of the variable itself
  - tells the scanf_s() where to find the variable to store the new value

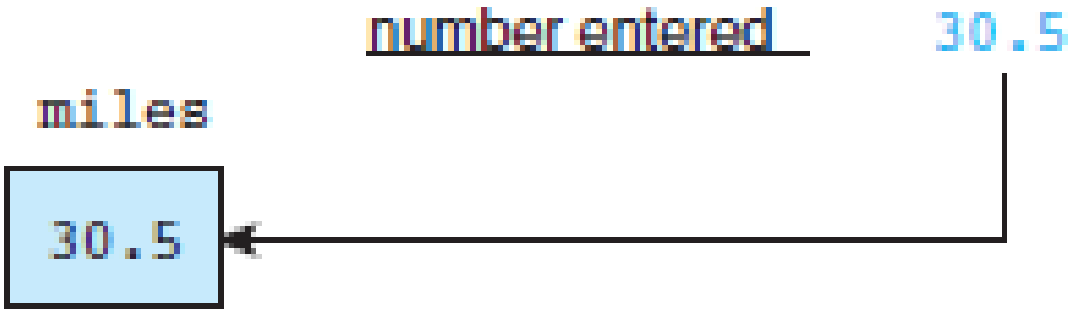
The scanf_s Function

- Copies data from the standard input device (usually the keyboard) into a variable.
- General format:
  - scanf_s(“Format string”, &variable);

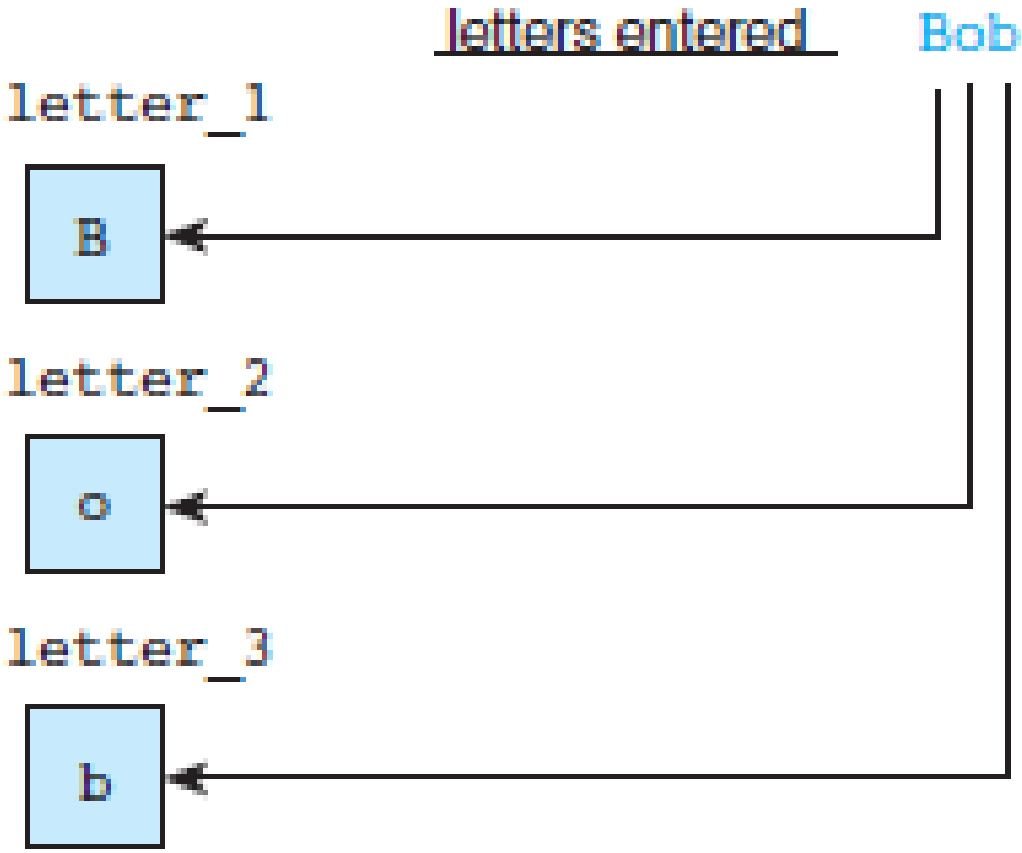
```
scanf_s(“%lf”, &miles);

scanf_s(“%c%c%c”, &letter_1, &letter_2, &letter_3);
```

Effect of scanf_s("%lf", &miles);



Scanning Data Line input Bob



The scanf_s function cont…

- Example :

```
int age;
printf("Enter your age: ");
scanf_s("%d", &age);
```

- Common Conversion Identifier used in printf and scanf_s functions.

	printf	scanf_s
int	%d	%d
float	%f	%f
double	%f	%lf
char	%c	%c
string	%s	%s

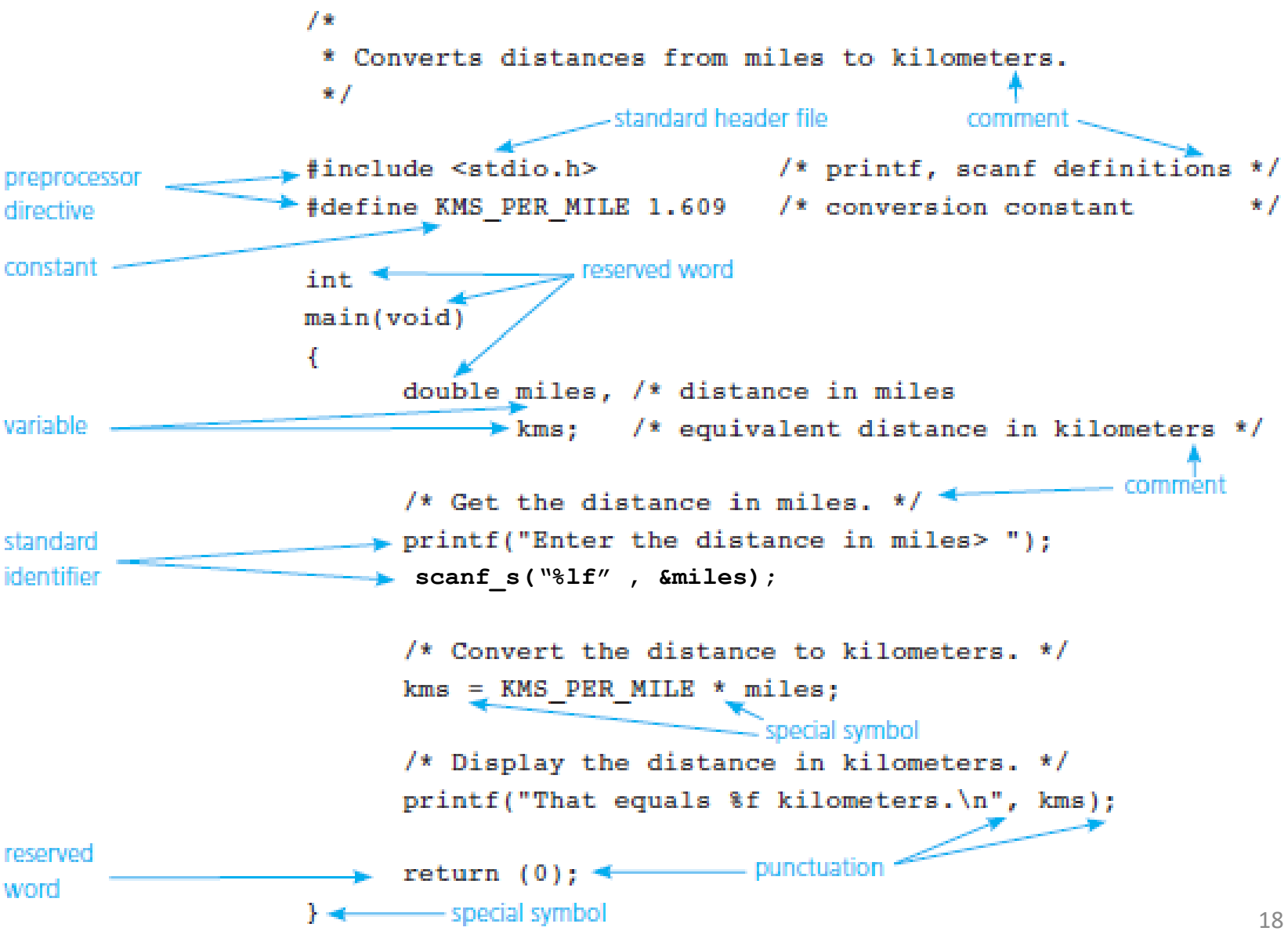
The scanf_s function cont…

- If you want the user to enter more than one value, you serialise the inputs.
- Example:

```
float height, weight;

printf("Please enter your height and weight:");
scanf_s("%f%f", &height, &weight);
```

Miles-to-Kilometers Conversion Program



getchar() and putchar()

- getchar() - read a character from standard input
- putchar() - write a character to standard output
- Example:

```
#include <stdio.h>
void main(void)
{
  char my_char;
  printf("Please type a character: ");
  my_char = getchar();
  printf("\nYou have typed this character: ");
  putchar(my_char);
}
```

getchar() and putchar()

- Alternatively, you can write the previous code using normal scanf and %c placeholder.
- Example

```
#include <stdio.h>
void main(void)
{
  char my_char;
  printf(“Please type a character: ”);
  scanf_s(“%c”,&my_char);
  printf(“\nYou have typed this character: %c ”, my_char);
}
```

Few notes on C program

- C is ***case-sensitive***
  - Word, word, WorD, WORD, WOrD, worD, etc are all different variables / expressions

    Eg. sum = 23 + 7
    - What is the value of Sum after this addition ?

- Comments
  - are inserted into the code using /* to start and */ to end a comment
  - Some compiler support comments starting with '//'
  - Provides supplementary information but is ignored by the preprocessor and compiler
    - `/* This is a comment */`
    - `// This is a comment too`

Few notes on C program

- Reserved Words
  - Keywords that identify language entities such as statements, data types, language attributes, etc.
  - Have special meaning to the compiler, cannot be used as identifiers (variable, function name) in our program.
  - Should be typed in lowercase.
  - Example: const, double, int, main, void,printf, while, for, else (etc..

Few notes on C program

- Punctuators (separators)
  - Symbols used to separate different parts of the C program.
  - These punctuators include:

    [ ] ( ) { } , ; “: * #
  - Usage example:

```
void main (void)
{
    int num = 10;
    printf (“% d”,num);
}
```

Common Programming Errors

- ***Debugging*** → Process removing errors from a program

- Three (3) kinds of errors :
  - Syntax Error
    - a violation of the C grammar rules, detected during program translation (compilation).
    - statement cannot be translated and program cannot be executed

Common Programming Errors

- Run-time errors
  - An attempt to perform an invalid operation, detected during program execution.
  - Occurs when the program directs the computer to perform an illegal operation, such as dividing a number by zero.
  - The computer will stop executing the program, and displays a diagnostic message indicates the line where the error was detected

Common Programming Errors

- Logic Error/Design Error
  - An error caused by following an incorrect algorithm
  - Very difficult to detect - it does not cause run-time error and does not display message errors.
  - The only sign of logic error – incorrect program output
  - Can be detected by testing the program thoroughly, comparing its output to calculated results
  - To prevent – carefully desk checking the algorithm and written program before you actually type it

Break 10 mins

Operators

- C supports a rich set of built-in operators
- Operators are used in programs to manipulate data and variables
- C operators can be classified into a number of categories
  1. Arithmetic operators
  2. Relational operators
  3. Logical operators
  4. Assignment operators
  5. Increment and decrement operators
  6. Conditional operators
  7. Bitwise operators
  8. Special operators

Arithmetic Operators

Arithmetic Operators

Operator	*Meaning*
+	Addition or unary plus
–	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

- The operators all work the same way as they do in other languages
- These can operate on any built-in data type allowed in C

Arithmetic Operators

- When both the operands in a single arithmetic expression such as a+b are integers
- Integer arithmetic always yields an integer value
- Ex) a = 14 , b = 4

  a – b = 10
  a + b = 18
  a * b = 56
  a / b = 3 (decimal part truncated)
  a % b = 2 (remainder of division)

Arithmetic Operators

- During integer division, if both the operands are of the same sign, the result is truncated towards zero

6 / 7 = 0 and -6 / -7 = 0

But -6 / 7 = 0 or -1 (Machine dependent)

- Similarly, during modulo division, the sign of the result is always the sign of the first operand (the dividend)

-14 % 3 = -2

-14 % -3 = -2

14 % -3 = 2

Example Program

Program

```
main ()
{
   int months, days ;

   printf("Enter days\n") ;
   scanf("%d", &days) ;

   months = days / 30 ;
   days = days % 30 ;
   printf("Months = %d Days = %d", months, days) ;
}
```

Output

```
Enter days
265
Months = 8 Days = 25
Enter days
364
Months = 12 Days = 4
Enter days
45
Months = 1 Days = 15
```

Illustration of integer arithmetic

Arithmetic Operators

- Real Arithmetic

  if x , y and z are floats

  x = 6.0 / 7.0 = 0.857143
  y = 1.0 / 3.0 = 0.333333
  z = -2.0 / 3.0 = -0.666667

- Mixed-mode Arithmetic

  one of the operands is real and the other is integer

  15 / 10.0 = 1.5 (mixed)
  15 / 10 = 1 (unmixed)

Relational Operators

Relational Operators

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

- Compare two quantities and depending on their relation, take certain decisions.

- These comparisons can be done with the help of *relational operators*

Relational Operators

- Arithmetic operators have a higher priority over relational operators
- Relational expression are used in *decision statements* such as **if** and **while**

Relational Operator Complements

Among the six relational operators, each one is a complement of another operator.

>	is complement of	<=
<	is complement of	>=
==	is complement of	!=

We can simplify an expression involving the *not* and the *less than* operators using the complements as shown below:

Actual one	*Simplified one*
!(x < y)	x >= y
!(x > y)	x <= y
!(x != y)	x == y
!(x <= y)	x > y
!(x >= y)	x < y
!(x == y)	x != y

Logical Operators

- C has the following three *logical operators*

  && meaning AND

  || meaning OR

  ! meaning NOT

- The logical operators && and || are used when we want to test more than one condition and make decisions

Truth Table

op-1	*op-2*	*Value of the expression*	
		op-1 && op-2	*op-1 || op-2*
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

Logical Operators

- Some examples of the usage of logical expressions are:
  1. if (age > 55 && salary < 1000)
  2. if (number < 0 || number > 100)

- Relative precedence of the relational and logical operators is as follows :

Highest	!
	> >= < <=
	== !=
	&&
Lowest	||

Assignment Operators

Shorthand Assignment Operators

Statement with simple assignment operator	*Statement with shorthand operator*
a = a + 1	a += 1
a = a – 1	a –= 1
a = a * (n+1)	a *= n+1
a = a / (n+1)	a /= n+1
a = a % b	a %= b

- Shorthand assignment operators has three advantages

1. What appears on the left-hand side need not be repeated and therefore it becomes easier to write.

2. The statement is more concise and easier to read.

3. the statement is more efficient.

Example Program

Program

```
#define   N  100
#define   A  2
main()
{
      int a;
      a = A;
      while( a < N )
      {
            printf("%d\n", a);
            a *= a;
      }
}
```

Output

```
2
4
16
```

*Use of shorthand operator *=*

Increment and Decrement Operators

```
++m; or m++;
--m; or m--;
```

++m; is equivalent to m = m + 1; (or m += 1;)
--m; is equivalent to m = m – 1; (or m -= 1;)

- While ++m and m++ mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement.

Increment and Decrement Operators

- Prefix operator first adds 1 to the operand and then the result is assigned to the variable on left.

m = 5;

y = ++m;

Output y and m = 6

- Postfix operator first assigns the value to the variable on left and then increments the operand.

m = 5;

y = m++;

Output y = 5 and m = 6

Increment and Decrement Operators

Rules for ++ and -- Operators

- Increment and decrement operators are unary operators and they require variable as their operands.
- When postfix ++ (or --) is used with a variable in an expression, the expression is evaluated first using the original value of the variable and then the variable is incremented (or decremented) by one.
- When prefix ++ (or --) is used in an expression, the variable is incremented (or decremented) first and then the expression is evaluated using the new value of the variable.
- The precedence and associatively of ++ and -- operators are the same as those of unary + and unary -.

Conditional Operators

- A ternary operator pair " ?: " is available in C to construct conditional expressions of the form

```
exp1 ? exp2 : exp3
```

- The operator ?: works as follows :

  1. *exp1* is evaluated.

  2. If *exp1* is nonzero(true), then *exp2* is evaluated and becomes the value of the expression.

  3. If *exp1* is false, *exp3* is evaluated and its value becomes the value of the expression

Conditional Operators

- For example,

```
a = 10;
b = 15;
x = ( a > b ) ? a : b;
```

- This can be achieved using the *if..else* statements.

```
If ( a > b )
  x = a;
else
  x = b;
```

Bitwise Operators

Bitwise Operators

Operator	*Meaning*
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right

- C has a distinction of supporting special operators known as *bitwise opertors* for manipulation of data at bit level.
- These operators are used for testing the bits, of shifting them right or left.
- Bitwise operators may not be applied to **float** or **double**.

Special Operators

- C supports some special operators.

1. comma operator

2. **sizeof** operator

3. pointer operators (& and *)

4. member selection operator (. And ->)

Special Operators

- **The Comma Operator**

- The comma operator can be used to link the related expressions together.

- A comma-linked list of expressions are evaluated *left to right* and the value of *right-most* expression is the value of the monbined expression.

Value = (x = 10, y = 5, x+y);

Output 10 to **x** , 5 to **y**, 15 to **value**

Special Operators

- Since comma operator has the lowest precedence of all operators, the parentheses are necessary.

- Some applications of comma operator are :

In **for** loops:

```
for ( n = 1, m = 10, n <= m; n++, m++)
```

In **while** loops:

```
while ( c = getchar (), c != '10')
```

Exchanging values:

```
t = x, x = y, y = t;
```

Special Operators

- **The sizeof Operator**

- The **sizeof** is a compile time operator and, when used with an operand, it returns the number of bytes the operand occupies.

```
m=sizeof  (sum);
n =sizeof (long int);
k =sizeof (235L);
```

- The **sizeof** operator is normally used to determine the lengths of **arrays** and **structures** when their sizes are not known to the programmer.

- It is also to allocate memory space dynamically to variables during execution of a program.

Example Program

Program

```
main()
{
     int a, b, c, d;

     a = 15;
     b = 10;
     c = ++a - b;

     printf("a = %d b = %d c = %d\n",a, b, c);

     d = b++ +a;

     printf("a = %d b = %d d = %d\n",a, b, d);
     printf("a/b = %d\n", a/b);
     printf("a%%b = %d\n", a%b);
     printf("a *= b = %d\n" , a*=b);
     printf("%d\n", (c>d) ? 1 : 0);
     printf("%d\n", (c<d) ? 1 : 0);
}
```

Output

```
a = 16 b = 10 c = 6
a = 16 b = 11 d = 26
a/b = 1
a%b = 5
a *= b = 176
0
1
```

Further illustration of arithmetic operators

Arithmetic Expressions

- An arithmetic expression is a combination of variables, constants, and operators arranged as per the syntax of the language.

- C can handle any complex mathematical expressions.

Expressions

Algebraic expression	*C expression*
a x b - c	a * b - c
(m+n) (x+y)	(m+n) * (x+y)
$\left(\frac{ab}{c}\right)$	a * b/c
$3x^2+2x+1$	3 * x * x + 2 * x + 1
$\left(\frac{x}{y}\right) + c$	x/y+c

Evaluation of Expressions

```
variable = expression;
```

- When the statement is encountered, the expression is evaluated first and the result then replaces the previous value of the variable on the left-hand side.

- All variables used in the expression must be assigned values before evaluation is attempted.

```
x = a * b – c;
y = b / c * a;
z = a – b / c + d;
```

- When these statements are used in a program, the variables a,b,c and d must be defined before they are used in the expressions.

Example Program

Program

```
main()
{
      float a, b, c, x, y, z;

      a = 9;
      b = 12;
      c = 3;

      x = a – b / 3 + c * 2 - 1;
      y = a – b / (3 + c) * (2 - 1);
      z = a – (b / (3 + c) * 2) - 1;

      printf("x = %f\n", x);
      printf("y = %f\n", y);
      printf("z = %f\n", z);
}
```

Output

```
x = 10.000000
y = 7.000000
z = 4.000000
```

Illustrations of evaluation of expressions

Precedence of Arithmetic Operators

- An arithmetic expression without parentheses will be evaluated from *left to right* using the rules of precedence of operators.

  High priority * / %

  Low priority + -

- Ex)

  $$x = a - b/3 + c * 2 - 1$$

  When a = 9, b = 12, and c = 3

  $$x = 9 - 12 /3 + 3 * 2 -1$$

Precedence of Arithmetic Operators

- First pass

  Step1 : x = 9 – 4 + 3 * 2 -1

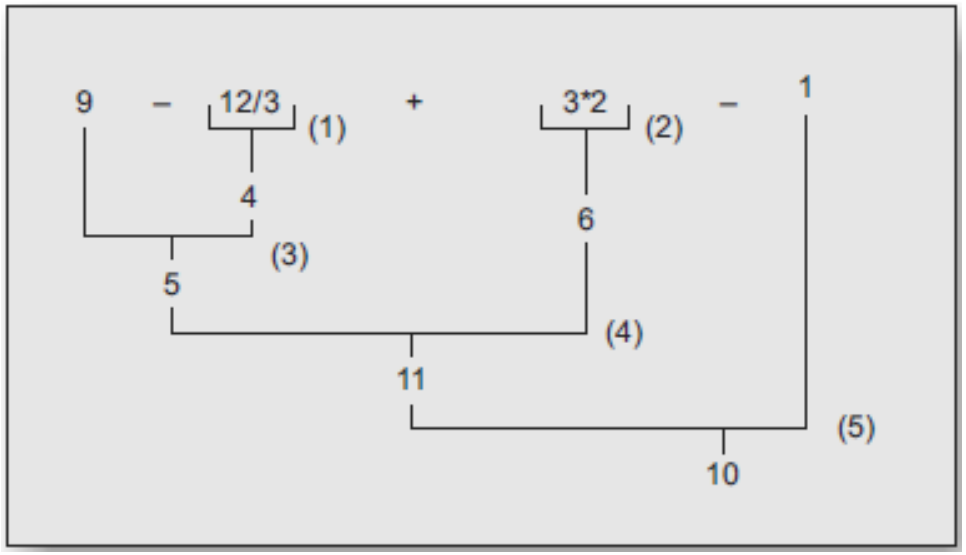
  Step2 : x = 9 – 4 + 6 -1

- Second pass

  Step3 : x = 5 + 6 – 1

  Step4 : x = 11 – 1

  Step5 : x = 10



Precedence of Arithmetic Operators

- Consider the same expression with parentheses as shown below :

  9 – 12 / (3 + 3) * (2 – 1)

- First pass

  Step1 : 9 – 12 / 6 * (2 – 1)

  Step2 : 9 – 12 / 6 * 1

- Second pass

  Step3 : 9 – 2 * 1

  Step4 : 9 – 2

- Third pass

  Step5 : 7

Precedence of Arithmetic Operators

Rules for Evaluation of Expression

- First, parenthesized sub expression from left to right are evaluated.
- If parentheses are nested, the evaluation begins with the innermost sub-expression.
- The precedence rule is applied in determining the order of application of operators in evaluating sub-expressions
- The associativity rule is applied when two or more operators of the same precedence level appear in a sub-expression.
- Arithmetic expressions are evaluated from left to right using the rules of precedence.
- When parentheses are used, the expressions within parentheses assume highest priority.

Some Computational Problems

- We know that the computer gives approximate values for real numbers and the errors due to such approximations may lead to serious problems.

```
a = 1.0 / 3.0;
b = a * 3.0;
```

- There is no guarantee that the value of **b** computed in a program will equal 1.
- Another problem is division by zero. On most computers, any attempt to divide a number by zero will result in abnormal termination of the program
- The third problem is to avoid overflow of underflow errors. It is our responsibility to guarantee that operands are of the correct type and range, and the result may not produce any overflow or underflow.

Some Computational Problems
Example Program

Program

```
/*---------- Sum of n terms of 1/n ----------*/
   main()
   {
        float sum, n, term ;
        int count = 1 ;

        sum = 0 ;
        printf("Enter value of n\n") ;
          scanf("%f", &n) ;
        term = 1.0/n ;
        while( count <= n )
        {
             sum = sum + term ;
             count++ ;
        }
        printf("Sum = %f\n", sum) ;
   }
```

Self study

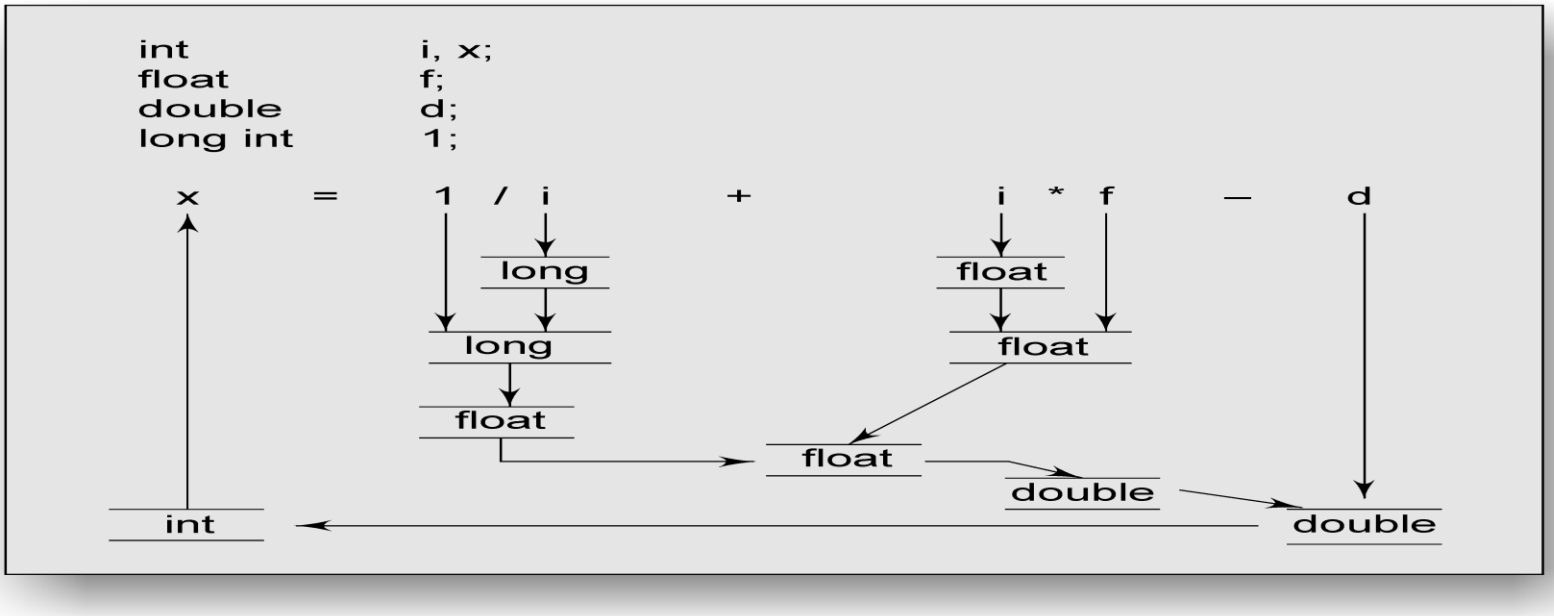
Output

```
   Enter value of n
   99
   Sum = 1.000001
   Enter value of n
   143
   Sum = 0.999999
```

Round-off errors in floating point computations

Type Conversions in Expressions

- **Implicit Type Conversion**
- C permits mixing of constants and variables of different types in an expression.
- If the operands are of different types, the ‘lower’ type is automatically converted to the ‘higher’ type before the operation proceeds



Type Conversions in Expressions

- All **short** and **char** are automatically converted to **int**; then

  1. if one of the operands is **long double**, the other will be converted to **long double** and the result will be **long double**;

  2. else, if one of the operands is **double**, the other will be converted to **double** and the result will be **double**;

  3. else, if one of the operands is **float**, the other will be converted to **float** and the result will be **float**;

  4. else, if one of the operands is **unsigned long int**, the other will be converted to **unsigned long int** and the result will be **unsigned long int;**

Type Conversions in Expressions

5. else, if one of the operands is **long int** and the other is **unsigned int**,
   (a) if **unsigned int** can be converted to **long int**, the **unsigned int** operand will be converted as such and the result will be **long int**;
   (b) else, both operands will be converted to **unsigned long int** and the result will be **unsigned long int**;

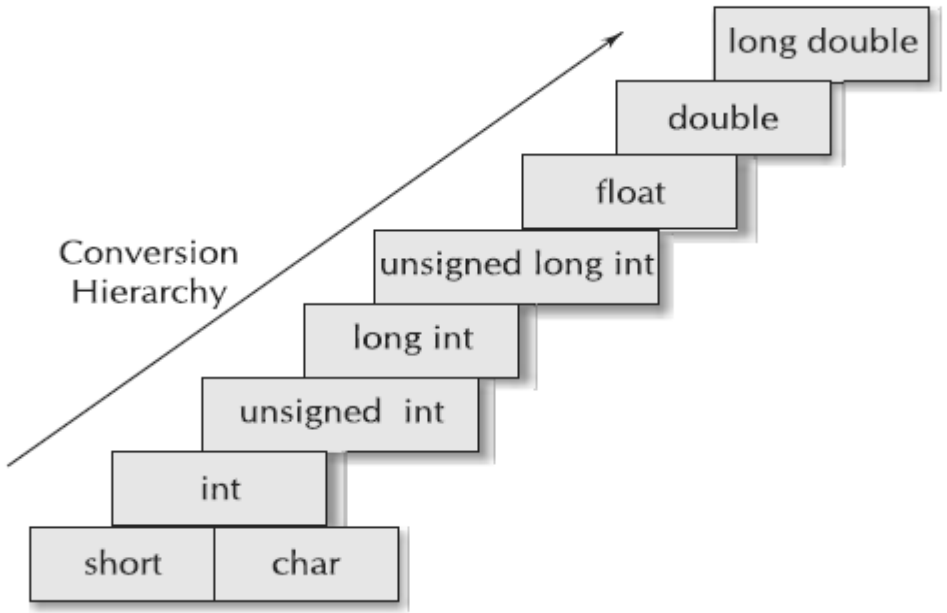
6. else, if one of the operands is **long int**, the other will be converted to **long int** and the result will be **long int**;

7. else, if one of the operands is **unsigned int**, the other will be converted to **unsigned int** and the result will be **unsigned int**.

Type Conversions in Expressions

Conversion Hierarchy

Note that, C uses the rule that, in all expressions except assignments, any implicit type conversions are made from a lower size type to a higher size type as shown below:



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

1. **float** to **int** causes truncation of the fractional part.

2. **double** to **float** causes rounding of digits.

3. **long int** to **int** causes dropping of the excess higher order bits.

Type Conversions in Expressions

- **Explicit Conversion**

(type-name)*expression*

Ex) ratio = female_number / male_number

- Since **female_number** and **male_number** are declared as 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 one of the variables to the floating point.

ratio = (**float**) female_number / male_number

Type Conversions in Expressions

- Note that in no way does the operator (**float**) affect the value of the variable **female number**.

- And also, the type of **female number** remains as **int** in the other parts of the program.

Use of Casts

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.

Example Program

Program

```
main()
{
      float  sum ;
      int    n ;

      sum = 0 ;

      for( n = 1 ; n <= 10 ; ++n )
      {
         sum = sum + 1/(float)n ;
         printf("%2d %6.4f\n", n, sum) ;
      }
}
```

Output

```
 1  1.0000
 2  1.5000
 3  1.8333
 4  2.0833
 5  2.2833
 6  2.4500
 7  2.5929
 8  2.7179
 9  2.8290
10  2.9290
```

Use of a cast

Operator Precedence and Associativity

- The operators at the higher level of precedence are evaluated first.
- The operators of the same precedence are evaluated either from ‘left to right’ or from ‘right to left’, depending on the level

Summary of C Operators

Operator	*Description*	*Associativity*	*Rank*
( ) [ ]	Function call Aray element reference	Left to right	1
+ – ++ – – ! ~ * & sizeof (type)	Unary plus Unary minus Increment Decrement Logical negation Ones complement Pointer reference (indirection) Address Size of an object Type cast (conversion)	Right to left	2
* / %	Multiplication Division Modulus	Left to right	3
+ –	Addition Subtraction	Left to right	4

Operator Precedence and Associativity

<<	Left shift	Left to right	5
>>	Right shift		
<	Less than	Left to right	6
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
==	Equality	Left to right	7
!=	Inequality		
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
|	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
||	Logical OR	Left to right	12
?:	Conditional expression	Right to left	13
=	Assignment operators	Right to left	14
* = /= %=			
+= -= &=			
^= |=			
<<= >>=			
,	Comma operator	Left to right	15

Mathematical Functions

- Most of the C compilers support these basic math functions : < math.h >

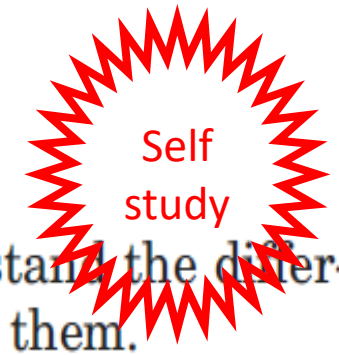
- Note : 1. **x** and **y** should be declared as **double**.
  2. In trigonometric and hyperbolic functions, **x** and **y** are in radians.
  3. All the functions return a **double.**
  4. C99 has added **float** and **long double** versions of these functions.
  5. C99 has added many more mathematical functions.
  6. See the Appendix “C99 Features” for details

Mathematical Functions

Math functions

Function	*Meaning*
Trigonometric	
acos(x)	Arc cosine of x
asin(x)	Arc sine of x
atan(x)	Arc tangent of x
atan 2(x,y)	Arc tangent of x/y
cos(x)	Cosine of x
sin(x)	Sine of x
tan(x)	Tangent of x
Hyperbolic	
cosh(x)	Hyperbolic cosine of x
sinh(x)	Hyperbolic sine of x
tanh(x)	Hyperbolic tangent of x
Other functions	
ceil(x)	x rounded up to the nearest integer
exp(x)	e to the x power (e^x)
fabs(x)	Absolute value of x.
floor(x)	x rounded down to the nearest integer
fmod(x,y)	Remainder of x/y
log(x)	Natural log of x, x > 0
log10(x)	Base 10 log of x, x > 0
pow(x,y)	x to the power y (x^y)
sqrt(x)	Square root of x, x >= 0

Things to Remember

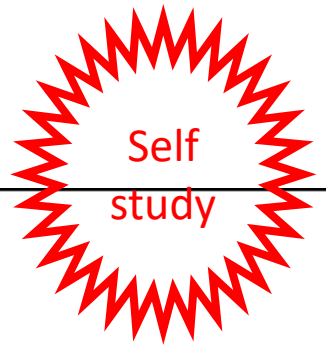


- Use *decrement* and *increment* operators carefully. Understand the difference between **postfix** and **prefix** operations before using them.
- Add parentheses wherever you feel they would help to make the evaluation order clear.
- Be aware of side effects produced by some expressions.
- Avoid any attempt to divide by zero. It is normally undefined. It will either result in a fatal error or in incorrect results.
- Do not forget a semicolon at the end of an expression.
- Understand clearly the precedence of operators in an expression. Use parentheses, if necessary.
- Associativity is applied when more than one operator of the same precedence are used in an expression. Understand which operators associate from right to left and which associate from left to right.
- Do not use *increment* or *decrement* operators with any expression other than a *variable identifier*.

Things to Remember

- It is illegal to apply modules operator % with anything other than integers.
- Do not use a variable in an expression before it has been assigned a value.
- Integer division always truncates the decimal part of the result. Use it carefully. Use casting where necessary.
- The result of an expression is converted to the type of the variable on the left of the assignment before assigning the value to it. Be careful about the loss of information during the conversion.
- All mathematical functions implement *double* type parameters and return *double* type values.
- It is an error if any space appears between the two symbols of the operators ==, !=, <= and >=.
- It is an error if the two symbols of the operators !=, <= and >= are reversed.
- Use spaces on either side of binary operator to improve the readability of the code.
- Do not use increment and decrement operators to floating point variables.
- Do not confuse the equality operator == with the assignment operator =.

Case Studies




```
Program
   #define BASE_SALARY  1500.00
   #define BONUS_RATE    200.00
   #define COMMISSION      0.02

   main()
   {
        int quantity ;
        float gross_salary, price ;
        float bonus, commission ;

        printf("Input number sold and price\n") ;
        scanf("%d %f", &quantity, &price) ;

        bonus          = BONUS_RATE * quantity ;
        commission     = COMMISSION * quantity * price ;
        gross_salary   = BASE_SALARY + bonus + commission ;

        printf("\n");
        printf("Bonus          = %6.2f\n", bonus) ;
        printf("Commission     = %6.2f\n", commission) ;
        printf("Gross salary = %6.2f\n", gross_salary) ;
   }
Output
     Input number sold and price
     5  20450.00
     Bonus                = 1000.00
     Commission           = 2045.00
     Gross salary         = 4545.00
```

Program of salesman's salary