

First Year Engineer in computer science

CHAPTER 2 :

Variables, Data Types, Operators, & Input/output

C Variables

2

What is a variable in C ?

Variable in C is a storage unit which sets a space in memory to hold a value and can take different values at different times during program execution.

A variable in C has an **identifier** (a name) and a **data type** (type of values the variable can take).

Variable declaration requires that you inform C of the variable's name and data type.

Variables should be defined before using them in the program.

C Variables

3

Rules to construct a valid variable name (identifier):

1. A variable name may consists of letters(a..z, A..Z), digits (0..9) and underscore (_) characters.
2. A variable name must begin with a letter. Some systems allow to start the variable name with an underscore.
3. ANSI standard recognizes a length of 31 characters for a variable name.
4. Uppercase and lowercase are significant. That is the variables **Volume**, **volume**, and **VOLUME** are different.
5. The variable name may not be a C reserved word (keyword).

C Variables

4

Examples of variable names:

Valid

StudentName

Total_Marks

_temp

G1INF2

Invalid

Student Name

price\$

2year

return

C Variables

5

C keywords :

- Keywords (also called reserved words) are those words whose meaning is already defined by compiler.
- There are 32 keywords in C:

auto	break	case	char
const	continue	default	do
double	enum	else	extern
float	for	goto	if
int	long	return	register
signed	short	static	sizeof
struct	switch	typedef	union
unsigned	void	volatile	while

Data types

6

- Data type is the type of data that a variable can store such as integer, floating, character etc .
- The program reserve memory space for the variable based on its data type.
- There are 4 types of data types in C language : basic, derived, enumeration, void

Data types

7

- Basic data types are arithmetic types and are integer-based and floating-point based. C language supports both signed and unsigned variables. The memory size of basic data types may change according to 32 or 64 bit operating system.
- Derived types: They include Pointers, Array, Structures, and Unions.
- Enumerated types: They are arithmetic types and they are used to define variables that can only be assigned certain discrete integer values throughout the program.
- Void type indicates that no value is available.

Data types

8

- Basic data types are arithmetic types and are integer-based and floating-point based. C language supports both signed and unsigned literals. The memory size of basic data types may change according to 32 or 64 bit operating system.
- Derived types: They include Pointers, Array, Structures, and Unions.
- Enumerated types: They are arithmetic types and they are used to define variables that can only be assigned certain discrete integer values throughout the program.
- Void type indicates that no value is available.

Basic data types

9

Integer Types

type	Storage size	Value range
char, signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 254
int	2 or 4 bytes	-32768 to 32767 or -2147483648 to 2147483647
unsigned int	2 or 4 bytes	0 to 65535 or 0 to 4294967295
short	2 bytes	-32768 to 32767
unsigned short	2 bytes	0 to 65535
long	4 or 8 bytes	-2147483648 to 2147483647 or ...
Unsigned long	4 or 8 bytes	0 to 4294967295 or ...

Basic data types

10

Floating-Point Types

type	Storage size	Value range	precision
float	4 bytes	1.2E-38 to 3.4E+38	6 decimal places
double	8 bytes	2.3E-308 to 1.7E+308	15 decimal places
long double	16 bytes		19 decimal places

Basic data types

11

To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator. The expressions **sizeof(type)** yields the storage size of the object or type in bytes.

Following is an example to get the size of int type on any machine:

```
#include <stdio.h>

int main()
{
    printf("Storage size for int : %d \n", sizeof(int));
    return 0;
}
```

Variable Definition in C:

12

- Variable definition means to tell the compiler where and how much to create the storage for the variable.
- The syntax of variable definition is as follows:

type VariableName ;

- type must be a valid C data type.

Example :

```
int time ;  
char c ;  
float distance;  
float speed;
```

Variable Definition in C:

13

- We can also define several variables of the same type in one line. Variables' identifiers are separated by commas.

example :

```
int a, b, c;
```

In this example, we are defining 3 variables a, b, and c of type int.

Assigning values to variables

14

- values can be assigned to variables using the assignment operator (=). The general format statement is :

Syntax : variable = constant;

example :

```
int number ;  
float speed ;  
...  
speed = 120,5;  
number = 10;
```

Variable initialization

15

- Variables can be initialized (assigned an initial value) in their declaration as follows:

type variable_name = value;

example :

```
float pi = 3.14 ;
```

```
char c = 'a';
```

C Constants (Literals)

16

- The constants refer to fixed values that the program may not change during its execution. These fixed values are also called literals.
- Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal.

C Constants (Literals)

17

Integer literals

- An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.
- An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Here are some examples of integer literals:

84	/* decimal (int) */
84u	/* decimal (unsigned int) */
84UL	/* decimal (unsigned long) */
0114	/* octal */
0x54	/* hexadecimal */
0X54L	/* hexadecimal (long)*/

C Constants (Literals)

18

Floating-point literals

- A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.
- While representing using exponential form, you must include the decimal point, the exponent, or both and while representing using decimal form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.

Here are some examples of floating-point literals:

3.14159 /* Legal */

314159E-5L /* Legal */

510E /* Illegal: incomplete exponent */

210f /* Illegal: no decimal or exponent */

.e55 /* Illegal: missing integer or fraction */

C Constants (Literals)

19

Character constants

Character literals are enclosed in single quotes, e.g., 'x' and can be stored in a simple variable of char type.

Escape characters or backslash characters

There are certain characters in C when they are preceded by a backslash they will have special meaning :

<code>\n</code>	newline	<code>\a</code>	alert (beep)
<code>\r</code>	carriage return	<code>\\</code>	backslash (\)
<code>\t</code>	horizontal tab	<code>\'</code>	single quote(')
<code>\v</code>	vertical tab	<code>\"</code>	double quote(“)
<code>\b</code>	backspace	<code>\?</code>	question mark (?)

C Constants (Literals)

20

String literals

A string constant is a sequence of characters enclosed in double quote, the characters may be letters, numbers, special characters and blank space etc

EX : "good" , "a" , "+123" , "1-/a" , "Earth is round\n"

Defining Constants

21

There are two ways in C to define constants: using **#define** preprocessor and using **const** keyword.

1. #define preprocessor

syntax: **#define** **identifier** **value**

examples :

```
#define PI 3.14
```

```
#define msg "finished"
```

1.

2. Using const keyword

syntax: **const** **type** **identifier** = **value** ;

example: const float PI = 3.14;

Now, the value of PI variable can't be changed.



Operators

Operators

23

Operators are symbols which take one or more operands and perform arithmetic or logical computations.

Operands are variables or constants which are used in conjunction with operators to evaluate the expression.

Expression is a combination of operands and operators that evaluates to a single value.

Operators

24

C language is rich in built-in operators and provides the following types of operators:

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Unary or Increment and decrement operators
- Bitwise operators
- Other operators

Operators : Arithmetic Operators

25

Following table shows all the **arithmetic operators** supported by C language.

operator	meaning	example
+	addition	$7 + 4$ gives 11
-	subtraction	$7 - 4$ gives 3
*	multiplication	$7 * 4$ gives 28
/	division	$7 / 4$ gives 1 $7 / 4.0$ gives 1,75
%	Modulo (remainder of integer division)	$7 \% 4$ gives 3

Operators : Relational Operators

26

A **relational operator** checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Operands may be variables, constants or expressions.

Operator	Meaning	Example	Return value
<	is less than	$2 < 9$	1
< =	is less than or equal to	$2 < = 2$	1
>	is greater than	$2 > 9$	0
> =	is greater than or equal to	$3 > = 2$	1
= =	is equal to	$2 = = 3$	0
!=	is not equal to	$2 != 2$	0

Operators : Logical Operators

27

These operators are used to combine the results of two or more conditions. An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false.

Operator	Meaning	Example	Return value
&&	Logical AND	(9>2)&&(17>2)	1
	Logical OR	(9>2) (17 == 7)	1
!	Logical NOT	!(9 == 9)	0

Logical AND: If any one condition false the complete condition becomes false.

Logical OR: If any one condition true the complete condition becomes true.

Logical Not: This operator reverses the value of the expression it operates on i.e, it makes a true expression false and false expression true.

Operators : Assignment Operators

28

Assignment operator (=) is used to assign a value or an expression or a value of a variable to another variable.

Example : $x = 5;$
 $y = x + 3;$

C provides compound assignment operators to assign a new value to variable after performing a specified operation.

Operator	Example	Meaning
$+=$	$x += y$	$x = x + y$
$-=$	$x -= y$	$x = x - y$
$*=$	$x *= y$	$x = x * y$
$/=$	$x /= y$	$x = x / y$
$\% =$	$x \% = y$	$x = x \% y$

Operators : Unary Operators

29

Unary operators (or Increment and Decrement Operators) operate on a single operand by incrementing (adding 1) or decrementing (subtracting 1) to a variable.

These operators operate only on variables.

syntax: ++variable; //increment operator
 --variable; //decrement operator

We have two types: Pre-Increment (Pre-Decrement) and Post-Increment (Post-Decrement) Operators.

++x : Pre increment, first increment and then do the operation.
--x : Pre decrement, first decrements and then do the operation.
x++ : Post increment, first do the operation and then increment.
x-- : Post decrement, first do the operation and then decrement.

Operators : Unary Operators

30

`++x` : Pre increment, first increment and then do the operation.
`--x` : Pre decrement, first decrements and then do the operation.
`x++` : Post increment, first do the operation and then increment.
`x--` : Post decrement, first do the operation and then decrement.

Ex: `let x = 5 ;`
`y = ++x ;`

`y = x++ ;`

Operators : Unary Operators

31

`++x` : Pre increment, first increment and then do the operation.
`--x` : Pre decrement, first decrements and then do the operation.
`x++` : Post increment, first do the operation and then increment.
`x--` : Post decrement, first do the operation and then decrement.

Ex: `let x = 5 ;`

<code>y = ++x ;</code>	<code>// is the same as</code>	<code>x = x + 1;</code>
	<code>//</code>	<code>y = x;</code>

`y = x++ ;`

Operators : Unary Operators

32

`++x` : Pre increment, first increment and then do the operation.
`--x` : Pre decrement, first decrements and then do the operation.
`x++` : Post increment, first do the operation and then increment.
`x--` : Post decrement, first do the operation and then decrement.

Ex: let `x = 5 ;`

```
y = ++x ;    // is the same as    x = x + 1;
              //                  y = x;
              // result           x = 6 and y = 6
```

```
y = x++ ;
```


Operators : Unary Operators

33

`++x` : Pre increment, first increment and then do the operation.
`--x` : Pre decrement, first decrements and then do the operation.
`x++` : Post increment, first do the operation and then increment.
`x--` : Post decrement, first do the operation and then decrement.

Ex: let `x = 5 ;`

```
y = ++x ;    // is the same as    x = x + 1;
              //                    y = x;
              // result             x = 6 and y = 6
```

```
y = x++ ;    // is the same as    y = x ;
              //                    x = x + 1;
```

Operators : Unary Operators

34

`++x` : Pre increment, first increment and then do the operation.
`--x` : Pre decrement, first decrements and then do the operation.
`x++` : Post increment, first do the operation and then increment.
`x--` : Post decrement, first do the operation and then decrement.

Ex: let `x = 5 ;`

```
y = ++x ;    // is the same as    x = x + 1;
              //                      y = x;
              // result              x = 6 and y = 6
```

```
y = x++ ;    // is the same as    y = x ;
              //                      x = x + 1;
              // result              x = 6 and y = 5
```

Operators : Other Operators

35

Comma Operator: The comma operator is used to separate the statement elements such as variables, constants or expressions, and this operator is used to link the related expressions together, such expressions can be evaluated from left to right and the value of right most expressions is the value of combined expressions.

Ex : `val = (a=3, b=9, c=7, a+c) ;`

First assigns the value 3 to a, then assigns 9 to b, then assigns 7 to c, and finally 10 (3+7) to val.

Operator Precedence / Associativity

36

- Every operator has a **precedence** value. An expression containing more than one operator is known as complex expression. Complex expressions are executed according to precedence of operators.
- **Associativity** specifies the order in which the operators are evaluated with the same precedence in a complex expression. Associativity is of two ways, i.e left to right and right to left. Left to right associativity evaluates an expression starting from left and moving towards right. Right to left associativity proceeds from right to left.

Operator Precedence / Associativity

37

The precedence and associativity of operators in C.

operator	description	precedence	Associativity
()	parenthesis	1	left to right
++ -- !	Increment Decrement Not	2	right to left
* / %	Multiplication Division modulo	3	left to right
+ -	Addition subtraction	4	left to right
> >= < <=	Relational	5	Left to right

Operator Precedence / Associativity

38

The precedence and associativity of operators in C.

operator	description	precedence	Associativity
== !=	Equality Inequality	6	left to right
&&	Logical AND	7	left to right
	Logical OR	8	left to right
= *= /= %= += - =	Assignment	9	Right to left
,	Comma	10	Left to right

Operator Precedence / Associativity

39

Example : $a = 9 - 12 / 3 + 3 * 2 - 1 ;$

Operator Precedence / Associativity

40

Example : $a = 9 - 12 / 3 + 3 * 2 - 1 ;$

1st phase :

Operator Precedence / Associativity

41

Example : $a = 9 - 12 / 3 + 3 * 2 - 1 ;$

1st phase :

1 : $a = 9 - 4 + 3 * 2 - 1$

Operator Precedence / Associativity

42

Example : $a = 9 - 12 / 3 + 3 * 2 - 1 ;$

1st phase :

$$1 : a = 9 - 4 + 3 * 2 - 1$$

$$2 : a = 9 - 4 + 6 - 1$$

Operator Precedence / Associativity

43

Example : $a = 9 - 12 / 3 + 3 * 2 - 1 ;$

1st phase :

$$1 : a = 9 - 4 + 3 * 2 - 1$$

$$2 : a = 9 - 4 + 6 - 1$$

2nd phase :

Operator Precedence / Associativity

44

Example : $a = 9 - 12 / 3 + 3 * 2 - 1 ;$

1st phase :

$$1 : a = 9 - 4 + 3 * 2 - 1$$

$$2 : a = 9 - 4 + 6 - 1$$

2nd phase :

$$1 : a = 5 + 6 - 1$$

Operator Precedence / Associativity

45

Example : $a = 9 - 12 / 3 + 3 * 2 - 1 ;$

1st phase :

$$1 : a = 9 - 4 + 3 * 2 - 1$$

$$2 : a = 9 - 4 + 6 - 1$$

2nd phase :

$$1 : a = 5 + 6 - 1$$

$$2 : a = 11 - 1$$

Operator Precedence / Associativity

46

Example : $a = 9 - 12 / 3 + 3 * 2 - 1 ;$

1st phase :

$$1 : a = 9 - 4 + 3 * 2 - 1$$

$$2 : a = 9 - 4 + 6 - 1$$

2nd phase :

$$1 : a = 5 + 6 - 1$$

$$2 : a = 11 - 1$$

$$3 : a = 10$$

Operator Precedence / Associativity

47

The order of evaluation can be changed by putting parenthesis in an expression.

Example : $a = 9 - 12 / (3 + 3) * (2 - 1) ;$

1st phase :

$$1 : a = 9 - 12 / \mathbf{6} * (2 - 1)$$

$$2 : a = 9 - 12 / 6 \mathbf{*} 1$$

2nd phase :

$$1 : a = 9 + \mathbf{2} * 1$$

$$2 : a = 9 + \mathbf{2}$$

3rd phase :

$$1 : a = \mathbf{11}$$



Input / Output

Input / Output

49

- **Input:** In any programming language input means to feed some data into program (read data) .
- **Output:** In any programming language output means to display some data on screen, printer or in any file.
- Several functions are available for input / output operations in “C”.
- Functions **printf()** and **scanf()** are the most commonly used to display out on the standard output (screen) and read from the standard input (keyboard) respectively.

Input / Output : printf()

50

- **printf()** used for formatted display on screen.

Syntax: **printf**(**format**, **arg1**, **arg2**, ...)

- **arg1**, **arg2**... are argument to display. An argument can be a constant, a variable, or an expression.
- format can contain literal strings as well as format specifiers (flags). The number of flags should be equal to the number of arguments to display.
- **flags** are of the form: **%[with][.precision]type**

Input / Output : printf()

51

- **printf()** used for formatted display on screen.

Syntax: **printf**(**format**, **arg1**, **arg2**, ...)

Input / Output : printf()

52

- **printf()** used for formatted display on screen.

Syntax: **printf**(**format**, **arg1**, **arg2**, ...)

Example :

```
int a = 20 ;
```

```
printf("the value of a is : %d", a ) ;
```

Input / Output : printf()

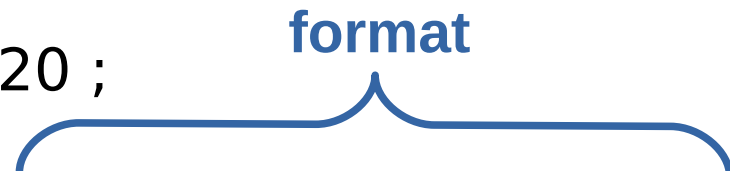
53

- **printf()** used for formatted display on screen.

Syntax: **printf**(**format**, **arg1**, **arg2**, ...)

Example :

```
int a = 20 ;  
printf("the value of a is : %d", a ) ;
```

A blue bracket is drawn above the string "the value of a is : %d" in the printf function call, with the word "format" written in blue above the bracket, indicating that this string is the format argument.

Input / Output : printf()

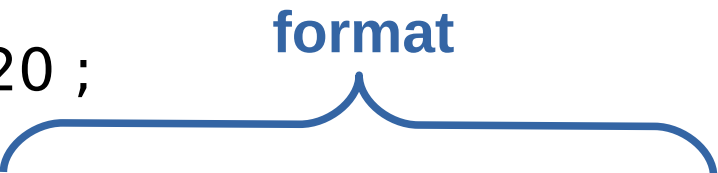
54

- **printf()** used for formatted display on screen.

Syntax: **printf**(**format**, **arg1**, **arg2**, ...)

Example :

```
int a = 20 ;  
printf("the value of a is : %d", a ) ;
```



literal strings



Input / Output : printf()

55

- **printf()** used for formatted display on screen.

Syntax: **printf**(**format**, **arg1**, **arg2**, ...)

Example :

```
int a = 20 ;  
printf( "the value of a is : %d", a ) ;
```

format

literal strings

flag

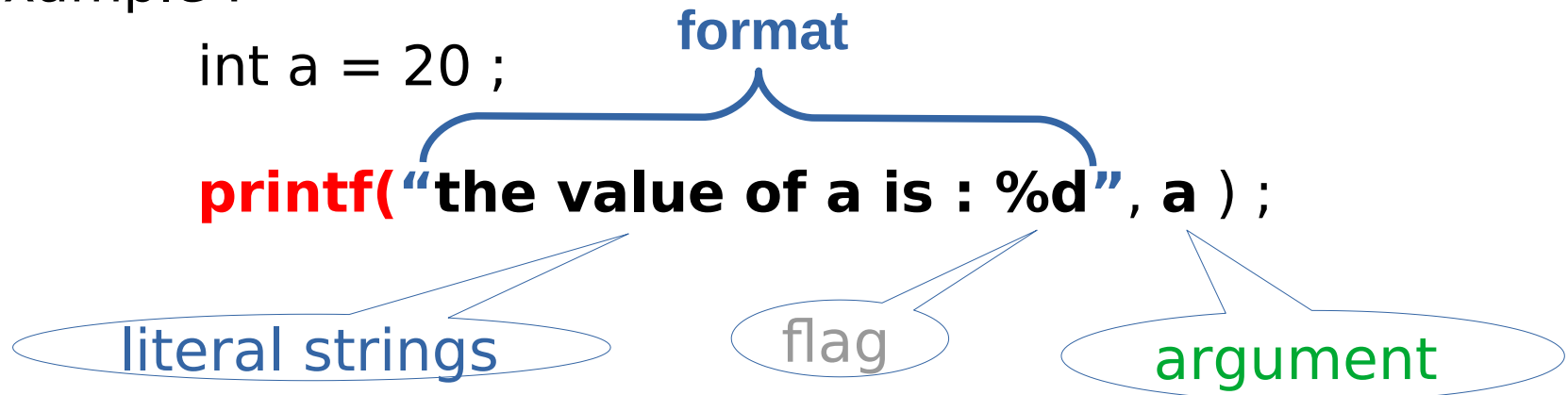
Input / Output : printf()

56

- **printf()** used for formatted display on screen.

Syntax: **printf**(**format**, **arg1**, **arg2**, ...)

Example :



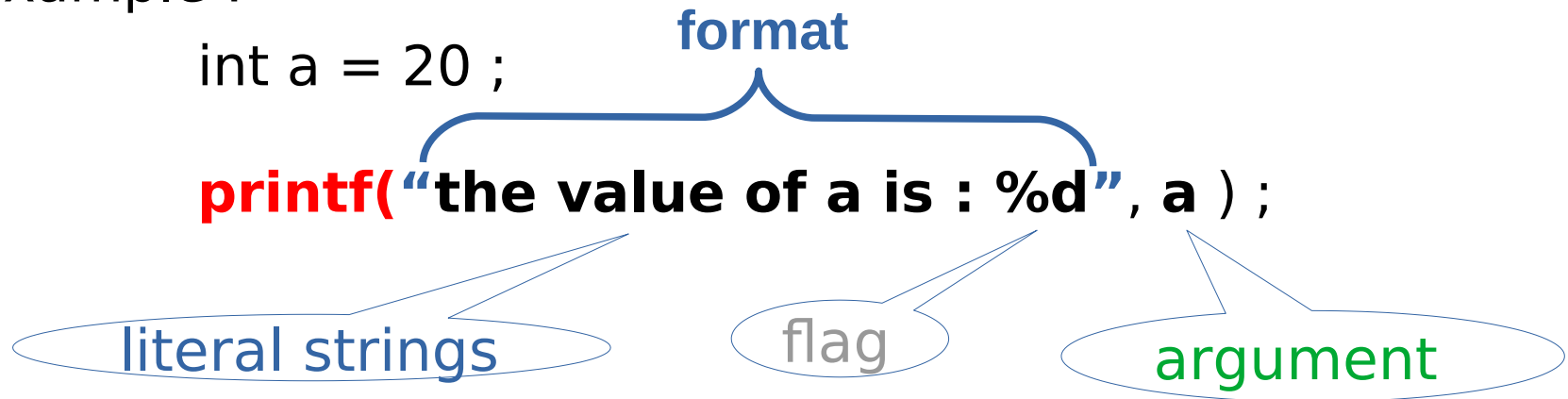
Input / Output : printf()

57

- **printf()** used for formatted display on screen.

Syntax: **printf**(**format**, **arg1**, **arg2**, ...)

Example :



output the value of a is : 20

Input / Output : printf()

58

- The **flags** are of the form: **%[with][.precision]type**
- The **type** for the flag can be one of the following:

type	Meaning	Examples
d or i	integer	printf ("%d",10); // prints 10
u	unsigned integer	printf ("%u",10); // prints 10
c	character	printf ("%c",'A'); // prints A
f	float	printf ("%f",2.3); // prints 2.3
e or E	float(exp)	1e3, 1.2E3, 1E-3
g	double	Printf("%g",2.3); // prints 2.3

Input / Output : printf()

59

- The **flags** are of the form: **%[with][.precision]type**
- The **with** for the flag:

example	output
<code>printf("%d",10);</code>	10
<code>printf("%4d",10);</code>	^ ^ 10 (^ is space)

- The **precision** is used to specify the number of decimals digits. Can be used with f or e flag.

example	output
<code>printf(" %.2f, %.0f", 1.141, 1.141);</code>	1.14, 1
<code>printf(" %.2e, %.0e", 1.141, 100.00);</code>	1.14e+00, 1e+02

Input / Output : scanf()

60

- **scanf()** used to read data from standard input (keyboard).

Syntax: **scanf(format, arg1, arg2, ...)**

- scanf() reads data from keyboard according to specified format
- The **format** contains flags (similar to printf). The number of flags should be equal to the number of arguments.
- Arguments have to be address of variables: **&variable**

Input / Output : scanf()

61

- **scanf()** used to read data from standard input (keyboard).

Syntax: **scanf(format, arg1, arg2, ...)**

- scanf() reads data from keyboard according to specified **format** (containing type flags).
- Arguments have to be address of variables : **&variable**
- The execution of the program is interrupted waiting for the user to introduce data.

examples:

```
scanf("%d", &x);    // read an integer value and put it in the
                    // variable x of type int.
```



Team: 1-ING-INF

twr5f8e