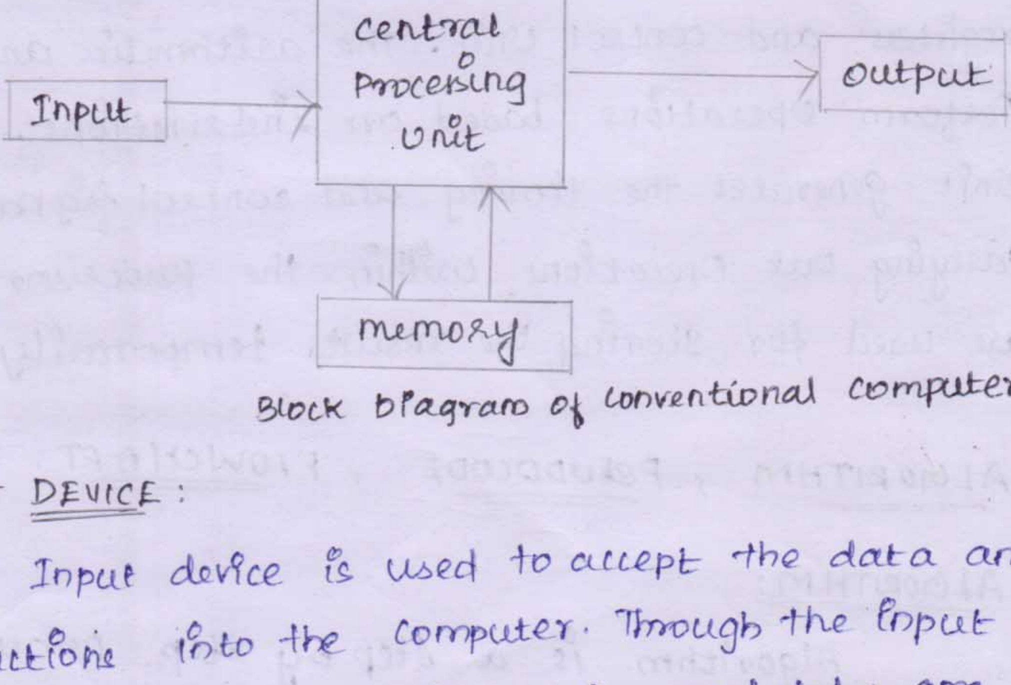
UNIT-1

INTRODUCTION TO C

Basic Blocks of computer:

Data to be processed by a computer appears in different forms, such as numeric, alphabetical characters either in uppercase or in lowercase, special characters and multimedia.

A computer is a programmable electronic machine that accepts instructions and data through input devices, manipulating data according to instructions and finally providing result to the output device. The result can be stored in memory or sent to the output device.



Block Diagram of conventional computer

INPUT DEVICE:

Input device is used to accept the data and instructions into the computer. Through the input device, i.e., keyboard, instructions and data are stored in computer memory.

OUTPUT DEVICE:

The output device is used to display the results on the screen or to send the data to an output device. The processed data is ultimately sent to the output device by the computer. The output device can be monitor, a printer, an LED, seven-segment display.

MEMORY:

Memory is used to store the program. There are two types of semiconductor memories. They are
i) RAM (Random access memory)
ii) ROM (Read only memory)

Central Processing Unit:

The heart of the computer system is a central Processing unit. It comprises arithmetic and logic unit, registers and control unit. The arithmetic and logic unit Perform operations based on instructions. The control unit generates the timing and control signals for carrying out operations within the processor. Registers are used for storing the results temporarily.

ALGORITHM, PSEUDOCODE, FLOWCHART:

ALGORITHM:

Algorithm is a step by step procedure, which defines a set of instructions to be executed which in a certain order to get the desired output.

Algorithms are generally created independent of underlying languages, i.e., an algorithm can be implemented in more than one programming language.

Characteristics of algorithm:

- unambiguous
- Input / output
- Finiteness
- Feasibility
- Independent

Example:

```
Step 1: Start
Step 2: Declare three integers a, b & c.
Step 3: Define values of a & b.
Step 4: add values of a & b
Step 5: Store output of Step 4 to c.
Step 6: Print c.
Step 7: Stop.
```

PSEUDOCODE:

A Pseudocode is an informal representation of an algorithm which is free from the programming language.

Pseudocode for factorial Number:

```
INPUT number
Set factorial := 1, i := 1
WHILE i <= number DO
    COMPUTE factorial := factorial * i
    INCREASE i by 1
END LOOP
PRINT factorial
```

FLOWCHART:

The flowchart is most widely used graphical representation of an algorithm and Procedural design workflows. It uses various symbols to show the operations and decisions to be followed in a program.

Types of flowchart:

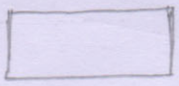
- Horizontal flowchart
- Panoramic flowchart
- vertical flowchart
- Architectural flowchart.

Symbols:

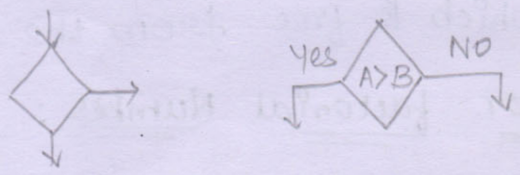
Terminal - Start / stop

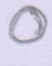
Input/output - Represent i/p of data and used to display o/p.

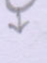
Process - Represent Arithmetic & data movement Instructions.

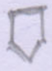


Decision - Represent Decision-making statements.



connector - Represent connection between Process. 

on-page connector - on-page Reference. 

Off-page connector - connect symbols in other page. 

STRUCTURE OF A 'C' PROGRAM:

Every c program contains a number of building blocks known as functions. Each function of it performs a specific task independently.

```
Include header file version

Global declaration section

/* comments */
int main (void)    /* function name */
{
   /* comments */
   Declaration part
   Executable part
   return 0;
}
user defined functions
{
   Body of the function
}
```

i) Include Header file section:

C program depends upon some header files for function definition that are used in the program. Each header file has extension '.h'. The header files are included at the beginning of the program in c language.

Example:

include <stdio.h> (or) # include "stdio.h"

ii) Global Declaration:

This section declares some variables that are used in more than one function. These variables are known as global variables. This section must be declared outside of all the functions.

iii) Function main():

Every program written in C must contain main() and its execution starts at the beginning of this function. In ASCII C standard, first line of C program from where program execution begins.

int main (void)

This is the function definition for main(). Parenthesis followed to main is to tell the user again that main() is a function. The int main(void) is a function that takes no arguments and returns a value of type int. Here in this line, int and void are keywords and they have special meanings assigned by the compiler. In case int is not mentioned in the above statement, by default the function returns an integer.

Alternately, one can also write the first line of C program from where program execution begins is as follows.

void main(void)

Here, the function takes no arguments and return nothing. Alternately, one can also write the same function as follows:

iv) Declaration part:

The declaration part declares the entire local variables that are used in executable part. Local variable scope is limited to that function where the local variables are declared. The initializations of variables can also be done in this section. The initialization means providing initial value to the variables.

v) Executable part:

This part contains the statements following the declaration of the variables. This part contains a set of statements or a single statement.

vi) user-defined function:

The functions defined by the user are called user-defined functions. These functions are defined outside the main() function.

v) Executable part:

This part contains the statements following the declaration of the variables. This part contains a set of statements or a single statement.

vi) user-defined Function:

The functions defined by the user are called user-defined functions. These functions are defined outside the main () function.

vii) Body of the Function:

The statements enclosed within the body of the function (between opening and closing brace) are called body of the function

comments:

comments are statements that give us information about the program which are to be placed between the delimeters /* and */. comments are not necessary in a program. However, to understand the flow of programs a programmer can insert comments in the program. Comments are to be inserted by the programmer. It is useful for documentation. The clarity of the program can be followed if it is properly documented.

Eg: /* This is comment */

DATATYPES:

All c compilers support a variety of data types. This enables the programmer to select the appropriate data type as per the need of the application. Generally, data is represented using numbers or characters. The numbers may be integer or real. The basic data types are char, int, float and double. c data type can be classified as follows.

i) Basic Data Type:

(a) Integer (int), (b) (character), (c) floating point (float), (d) double floating point (double) and (e) void.

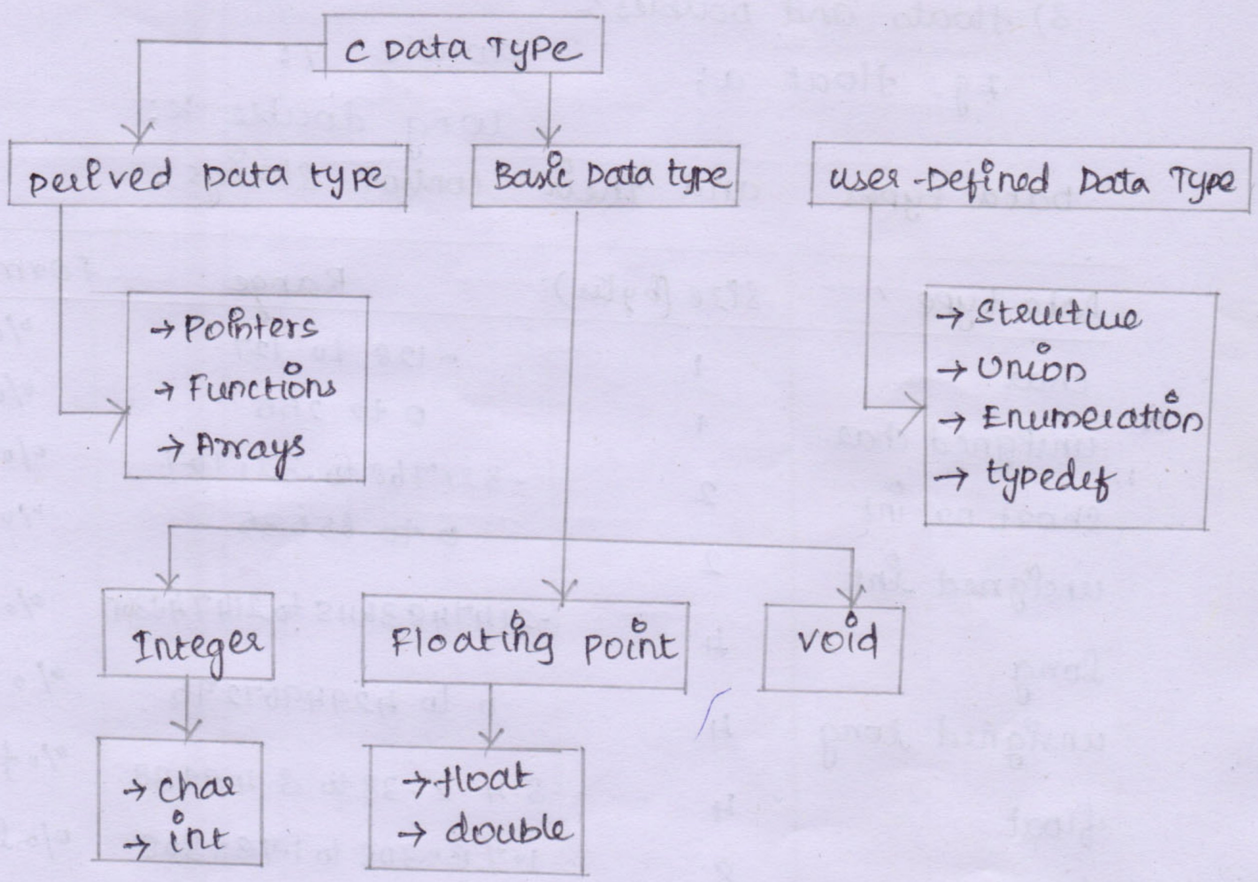
ii) Derived Data Type:

Derived data type are pointers, functions and arrays.

iii) user-defined Type:

struct, union and typedef are user-defined data types,

iv) void Data Type: This data type is explained in upcoming unit.



C DATA TYPES:

1) Integer Data Type:

a) int, short and long:

```
Eg:  int a=2;
     short int b=2;
     long b = 123456;
     long int c = 1234567;
```

b) Integers signed and unsigned:

```
Eg:  int a=2
     long int b=2;
     unsigned long b= 567898;
     unsigned short int c=223;
```

c) char signed and unsigned:

```
Eg:  char ch='b';     unsigned char ='b';
```

3) Floats and Doubles:

Eg: float a;   double y;

long double k;

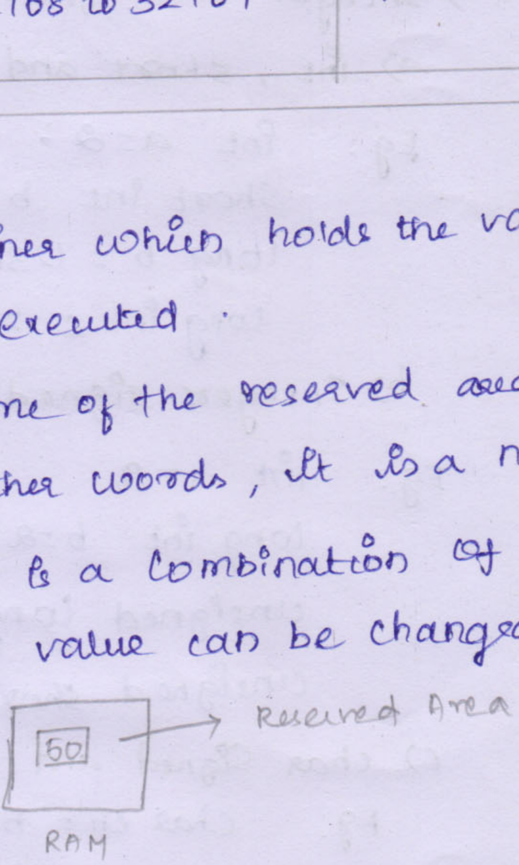
Data types and their control strings:

Data type	Size (bytes)	Range	Formal String
char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
short or int	2	-32,768 to 32,767	%i or %d
unsigned int	2	0 to 65535	%u
long	4	-2147483648 to 2147483647	%ld
unsigned long	4	0 to 4294967295	%lu
float	4	3.4 e-38 to 3.4 e+38	%f or %g
double	8	1.7 e-308 to 1.7e+308	%lf
long double	10	3.4 e-4932 to 1.1e+4932	%lf
enum	2*	-32768 to 32767	%d

VARIABLES:

A variable is a container which holds the value while the Java program is executed.

A variable is the name of the reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary+able" which means value can be changed.



int data = 50;

Syntax: Data-type variable-name;

TYPES of VARIABLE :

1) Local variable 2) Instance variable 3) static variable

1) Local variable :

A variable declared inside the body of the method is called local variable .

2) Instance variable :

A variable declared inside the class but outside the body of the method , is called an instance variable .

3) static variable :

A variable that is declared as static is called a static variable . It cannot be local . Memory allocation for static variable happens only once when the class is loaded in the memory .

INITIALIZING VARIABLES :

variables declared can be assigned or initialized using assignment Operator '='. The declaration and initialization can also be done in the same line .

Syntax :

```
variable_name = constant;
    or
data_type variable_name = constant;
```

Eg: x = 5 ; where x is an integer variable .

```
int y=4;
int x,y,z;
```

int x=y=z=3; /* invalid Statement */

$$\begin{tikzcd}
\boxed{\text{Data-type}} \arrow[d] & \boxed{\text{variable-Name}} \arrow[d] & \boxed{\text{Operator}} \arrow[d] & \boxed{\text{value}} \arrow[d] \\
\boxed{\text{int}} & \boxed{y} & \boxed{=} & \boxed{2}
\end{tikzcd}$$

value assignment

$$\begin{tikzcd}
\boxed{x} & = & \boxed{y} \arrow[ll, bend left] & = & \boxed{z} \arrow[ll, bend left] & = & \boxed{1} \arrow[ll, bend left]
\end{tikzcd}$$

DYNAMIC INITIALIZATION:

The initialization of variable at run time is called dynamic initialization.

Example:

```
void main()
{
    int r = 2;
    float area = 3.14 * r * r;
    clrscr();
    Printf ("Area = %g", area);
}
```

output: Area = 12.56.

TYPE CONVERSION:

Sometimes a programmer needs the result in a certain data type, for example division of 5 with 2 should return float value. practically, the compiler always return integer values both the arguments are of integer data type.

Example:

```
void main()
{ clrscr();
  printf("\n Division Operation Result");
  printf("\n Two Integers (5 & 2) : %d", 5/2);
  printf("\n One int & one float (5.5 & 2) : %g", 5.5/2);
  printf("\n Two Integers (5 & 2) : %g", (float)5/2);
```

```
Output:  Division Operation                 Result
         Two Integers (5 & 2)             :  2
         One int & one float (5.5 & 2)    :  2.75
         Two Integers (5 & 2)             :  2.5
```

Constant variable:

Variable remains unchanged during the program execution. The keyword const is then prefixed before the declaration.

Eg: const int n = 10;

Volatile variable:

The volatile variable are those variables that can be changed at any time by other external program or the same program.

Eg: volatile int d;

Constants, Operators:

CONSTANT:

constant is an entity in Programming that is immutable. In other words, the value that cannot be changed.

SYNTAX: Static final datatype identifier-name = value;

Eg: Static final double PRICE = 432.78;

Static and final Modifiers:

* The purpose to use the static modifier is to manage the memory.
* It also allows the variable to be available without loading any instance of the class in which it is defined
* The final modifier represents that the value of the variable cannot be changed. It also makes the primitive data type immutable or unchangeable.

OPERATORS:

An operator indicates an operation to be performed on data that yields a new value. Using various operators in c, one can link the variables and constants.

Types of Operator

Type of Operator	Symbolic Representation
Arithmetic Operators	+, -, *, / and %
Relational Operators	>, <, ==, >=, <= and !=

Types of Operator	Symbolic Representation
Logical operator	&&, || and !
Increment & decrement Operator	++ and – –
Assignment Operator	=, *=, /=, %=, +=, -=, &=, ^=, |=, <<=, >>=
Bitwise Operator	&, |, ^, >>, << and ~
comma Operator	,
conditional operators	? :

Properties of Operators:

i) Precedence: precedence refers to the priority given to the operator for a process. When an expression contains many operators, the operations are carried out according to the priority of the operators. The higher priority operations are solved first.

For Eg: In arithmetic Operators, the operators *, / and % are highest priority and of similar precedence. The operators + and - are having lowest precedence.

Example: 8+9 * 2 -10

The operator * is highest priority => 8 + 18 -10

Next, + and - have same priority. In such a situation, the left most operation is solved first.

= 26 -10

= 16

ii) Associativity:

When an expression has operators with equal precedence, the associativity property decides which operation to be carried out first. Associativity means the direction of execution.

a) Left to right: In this type, expression evaluation starts from the left to right direction.

Eg: 12 * 4 / 8 % 2
= 48 / 8 % 2
= 6 % 2
= 0

b) Right to left: In this type, expression evaluation starts from right to left direction.

Eg: x = 8+5%2
x = 8+1 = 9

Arithmetic Operator:

Operator	Description
+	Additive Operator
-	Subtraction Operator
*	Multiplication Operator
/	Division Operator
%	Remainder Operator

Example:

```
Class Arithmetic
{
  Public static void main (String [] args)
  {
    int result = 1+2;
    System.out.println("1+2 = " + result);
    int Original_result = result;
    result = result - 1;
    System.out.println(Original_result + " -1 = " + result);
    Original_result = result;
    result = result * 2;
    System.out.println(Original_result + " *2 = " + result);
    Original_result = result;
    result = result / 2;
    System.out.println(Original_result + " /2 = " + result);
    Original_result = result;
    result = result + 8;
    System.out.println(Original_result + " +8 = " + result);
    Original_result = result;
    result = result % 7;
    System.out.println(Orignal_result + " % 7 = " + result);
  }}
```

Unary Operator:

Unary Operator require only one Operand; they Perform various Operations.

Operator	Description
+	unary Plus Operator; Indicates Positive value.
–	unary minus Operator ; negates an expression
++	Increment Operator; Increments a value by 1
--	Decrement Operator; Decrements value by 1.
!	Logical complement Operator ; inverts the value of a boolean.

Relational Operator:

Relational Operators are used to compare numeric values. So these Operators are used to make conditions. Result of a relational expression produces boolean value as result.

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

Logical operators:

Logical Operators are also called Boolean operators. These are used to combine the results of one or more conditions. Result of a logical expression produces Boolean value as result.

Operators	Purpose
&	Logical AND
|	Logical OR
^	Logical XOR
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

Bitwise Operators:

Bitwise operators are used for bitwise manipulations. These do not work for float or boolean types.

Operators	Purpose	Operators	Purpose
		<=	Shift left
~	Bitwise unary NOT	&=	Bitwise AND assign
&	Bitwise AND	|=	Bitwise OR assign
|	Bitwise OR	^=	Bitwise XOR assign
^	Bitwise XOR	>>=	Shift right assign
>>	Shift right	>>>=	Shift right fill zero assign
>>>	Shift right zero fill	<<=	Shift left assign

INPUT and OUTPUT STATEMENTS :

In C Language input and output function are available in C compilee functions or C libraries provided with each C compilee implementation. These all functions are collectively known as standard I/O Library Function. Here I/O stands for Input and output used for different inputting and outputting statements. These I/O functions are categorized into three processing functions. console input/output function (deals with keyboard and monitor), disk input/output function (deals with floppy or hard disk) and port (input/output function - deals with a serial or parallel port).

$$\begin{tikzcd}
 & \text{Console Input/output (stdio.h)} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Standard input (Stdin)} & \text{standard output (Stdout)} & \text{Standard Error (Stderr)}
\end{tikzcd}$$

Stdin : This file is used to receive the input.

Stdout : This file is used to send or direct the output.

Stderr : This file used to display or store error messages.

There are mainly two of Input/output functions are used for this purpose. These are.

* Unformatted I/O functions
* Formatted I/O functions

unformatted I/o functions :

There are mainly six unformatted I/o functions:

* getchar( )
* Putchar( )
* gets( )
* puts( )
* getch( )
* getche( )

getchar( ):

This function is an Input function. It is used for reading a single character from the keyboard. It is a buffered function. Buffered functions get the input from the keyboard & store it in the memory buffer temporarily until you press the Enter key.

the general syntax is as:

v = getchar( ); ⇒

```
main( )
{ char n;
  n=getchar( );
}
```

Putchar( ):

This function is an output function. It is used to display a single character on the screen. The general syntax is as

put char (v); ⇒

```
main( )
{ char n;
  n = getchar( );
  putchar(n);
}
```

gets( ):

It will mark null character ('\0') in the memory at the end of the string when you press the enter key.

Syntax : gets(v);

Puts( ):

This function appends a newline "\n") character to the output.

Syntax : puts(v);

code:
```
main()
{
 char name[20];
 Puts("Enter the Name");
 gets(name);
 Puts("Name is : ");
 Puts(name);
}
```

getch( ):

The character data read by this function is directly assigned to a variable rather it goes to the memory buffer, the character data is directly assigned to a variable without the need to press the Enter key.

Syntax: v= getch();

Example:
```
main()
{
 char n;
 Puts("Enter the char");
 n= getch();
 Puts("char is : ");
 Putchar(n);
  getch();
}
```

getche( ):

All are same as getch( ) function except it is an echoed function. It means when you type the character data from the keyboard it will visible on the screen.

Syntax: v = getche( );

Code:
```
main ( )
{ char n;
  puts ("Enter the char");
  n = getche ( );
  puts ("char is : ");
  putchar(n);
  getche ( );
}
```

Formatted I/O Functions :

Formatted I/O function refers to an input or output data that has been arrange in a particular format.

* scanf ( )
* printf ( ) – output function

scanf( ):

The scanf( ) function is an input function. It is used to read the mixed type of data from keyboard. You can read integer, float and character data by using control codes or format codes.

Syntax: scanf ("control strings", arg1, arg2, ... argn);

printf( ) syntax: printf ("control strings", &v1, &v2, ... &vn);

Format code	Meaning
%c	To read a single character
%d	(short) To read a signed decimal integer
%ld	To read a signed long ~~decimal~~ integer
%e	To read a float value exponential
%f	To read short float value
%lf	To read long float value.
%g	To read double float value.
%h	To read short integer.
%i	To read an integer (decimal, octal, hexadecimal).
%o	To read an octal integer only.
%x	To read hexadecimal integer only.
%u	To read unsigned decimal integer.
%s	To read a string.
%[...]	To read a string of words from the defined range.
%[^]	To read string of words which are not from the defined range.