Unit-3

Input and Output

Conversion Specification

While the data is being output or input, it must be notified with some identifier and their formal specifier. Formal specifiers are the character starting with % sign and followed with a character. It specifies the type of data that is being processed. It is also called conversion specification.

Data types	**Format specifiers symbol**
integer	%d
unsigned integer	%u
octal	%o
Hexadecimal	%x
float (simple)	%f
float (exponential)	%e
character	%c
string	%s
long integer	%ld

Reading Input Data

Input data can be entered into the memory from a standard input device (keyboard). C Provides the scanf ( ) library function for entering input data. This function can take all types of values (numeric, character, string) as input. The scanf( ) function can be written as:

```
scanf ( "control string", address1, address2,…);
```

This function should have at least two parameters. First parameter is a control string which contains conversion specification characters, It should be within double quoted. The conversion specification characters maybe done or more; it depends on the number of variable we want to input. The other parameters are addresses or variables. In the scanf( ) function at least one address should be present. The address of a variable is found by preceding the variable name by an ampersand (&) sign, This sign is called the address operator and it gives the starting address of the variable name in the memory. A string variable is not preceded by & sign to get the address.

Some examples of scanf( ) function are as:

```
#include<stdio.h>

main ( )

{
  int marks;
  ………..
  scanf("%d", &marks);
  …………..
}
```

In this example, the control string contains only one conversion specification %d, which implies that one integer value should be entered as input. This entered value will be stored in the variable marks.

```
#include<stdio.h>
main ( )
{
  char ch;
  ………
  scanf("%c", &ch);
  …………
}
```

Here the control string contains conversion specification character %c, which means that a single character should be entered as input. This entered value will be stored in the variable ch.

```
#include<stdio.h>
main ( )
{
  float height;
  …………..
  scanf("%f", &height);
  …………….
}
```

Here the control string contains the conversion specification character %f, which means that a floating point number should be entered as input. This entered value will be stored in the variable height.

```
#include<stdio.h>
main ( )
```

```
{
  char str[30];
  ……………
  scanf("%s", str);
  ……………
}
```

In this example control string has conversion specification character %s implying that a string should be taken as input. Note that the variable str is not preceded by ampersand(&) sign. The entered string will be stored in the variable str.

Writing Output Data

Output data can be written from computer memory to standard output device (monitor) using printf ( ) library function. With this function all type of values (numeric, character or string) can be written as output. The printf( ) function can be written as:

```
printf("control string", variable1, variable2, ……….)
```

In this function the control string contains conversion specification characters and text. It should be enclosed within double quoted. The name of variables should not be preceded by an ampersand (&) sign. If the control string does not contain any conversion specification, then the variable names are not specified. Some example of printf( ) function are as:

```
#inlude<stdio.h>
main ( )
{
  printf("C is excellent\n");
}
```

Output:

```
C is excellent
```

Here control string has only text and no conversion specification character, hence the output is only text.

```
#include<stdio.h>
main( )
{
  int age;
  printf("Enter your age:");
  scanf("%d", &age);
}
```

Here also printf does not contain any conversion specification character and is used to display a message that tells the user to enter his/her age.

```
#include<stdio.h>
main( )
{
  int basic = 2000;
  ………..
  printf("%d", basic);
  ………….
}
```

In the above example control string contains a conversion specification character %d, which implies that an integer value will be displayed. The variable basic has the integer value which will be displayed as output.

```
#include<stdio.h>
main( )
{
  float height =5.6;
  ……………………
  printf("%f", height);
  ………………
}
```

Here control string has conversion specification %f, which means that floating point number will be displayed. The variable height has that floating point value which will be displayed as output.

```
#include<stdio.h>
main ( )
{
  char ch = '$';
  …………
  printf("%c", ch);
  ………..
}
```

In the above example, the control string has conversion specification character %c, means that single character will be displayed and variable ch has the character value.

```
#include<stdio.h>
main( )
{
  char str[30];
  …………..
printf("%s", str);
…………….
}
```

Here control string has conversion specification character %s, implying that a string will be displayed and variable name str is a character array, holding the string which will be displayed.

Formatted I/O

Formatted input and output means that data is entered and displayed in a particular format. Through format specification, better presentation of result can be obtained. Formats for different specifications are as:

Format for integer input

%wd

Here 'd' is the conversion specification character for integer value and 'w' is an integer number specifying the maximum field width of input data. If the length of input is more than this maximum field width then the values are not stored correctly. For example

scanf("%2d%3d",&a,&b);

i) When input data length is less than the given field width, then the input values are unaltered and stored in given variables.
   Input-
   6 39
   Result-
   6 is stored in a and 39 is stored in b.

ii) When input data length is equal to the given field width, then the input values are unaltered and stored in given variables.

Input-
27 489
Result-
27 is stored in a and 489 is stored in b.

iii) When input data length is more than the given field width, then the input values are altered and stored in the variable as-
Input-
278 3479
Result-
27 is stored in a and 8 is stored in b and the rest of input is ignored.

Format for integer output

%wd

Here w is the integer number specifying the minimum field width of the output data. If the length of the variable is less than the specified field width, then the variable is right justified with leading blanks.

For example-

printf("a=%3d, b=%4d",a,b);

i) When the length of variable is less than the width specifier.
Input-
78 9
Output-

a	=		7	8	,	b	=				9

The width specifier of first data is 3 while there are only 2 digits in it, so there is one leading blank. The width specifier of second data is 4 while there is only 1 digit, so there are 3 leading blanks.

ii) When the length of the variable is equal to the width specifier.
Input-
263 1941
Output-

a	=	2	6	3	,	b	=	1	9	4	1

iii) When the length of variable is more than the width specifier, then also the output is printed correctly.

Input-

2691 19412

Output-

a	=	2	6	9	1	,	b	=	1	9	4	1	2

Format for floating point numeric input

%wf

Here 'w' is the integer number specifying the total width of the input data (including the digits before and after decimal and the decimal itself).

For example-

scanf("%3f%4f",&x,&y);

i) When input data length is less than the given width, values are unaltered and stored in the variables.

Input-

5 5.9

Result-

5.0 is stored in x and 5.90 is stored in y.

ii) When input data length is equal to the given width, then the given values are unaltered and stored in the given variables.

Input-

5.3 5.92

Result-

5.3 is stored in x and 5.92 is stored in y.

ii) When input data length is more than the given width then the given values are altered and stored in the given variables as-

Input-

5.93 65.87

Result

5.9 is stored in x and 3.00 is stored in y.

Format for floating point numeric output

%w.nf

Here w is the integer number specifying the total width of the input data and n is the number of digits to be printed after decimal point. By default 6 digits are printed after the decimal.

For example-

printf(“x=%4.1f,y=%7.2f”,x,y);

value of variables-

8 5.9

Output:

x	=		8	.	0	,	y	=				5	.	9	0

Value of variables-

25.3 1635.92

Output

x	=	2	5	.	3	,	y	=	1	6	3	5	.	9	2

Value of variables-

15.231 65.875948

Output:

x	=	1	5	.	2	,	y	=			6	5	.	8	8

Format for string input

%ws

Here w specifies the total number of characters that will be stored in the string.

char str[8];

scanf(“%3s”,str);

Input-

programming

Only first three characters of this input will be stored in the string, so the characters in the string will be-

‘p’, ‘r’, ‘o’, ‘\0’

The null character ('\0') is automatically stored at the end.

Format for string output

%w.ns

Here w is the specified field width. Decimal point and 'n' are optional. If present then 'n' specifies that only first n characters of the string will be displayed and (w-n) leading blanks are displayed before string.

p	r	o	g	r	a	m	m	i	n	g

printf("%3s","programming");

					r	e	e	t	a

printf("%10s","reeta");

p	r	o

printf("%.3s","programming");

					p	r	o

printf("%8.3s","programming");