SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with ‘A+’ Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

PROGRAMMING FOR PROBLEM SOLVING

I YEAR - I SEM

UNIT 2 – C Programming Basics

TOPIC 7 – Managing Input Output Operations

INPUT/OUTPUT OPERATIONS

- Reading, processing, and writing of data are the **three** essential functions of a computer program.
- Most programs take some data as input and display the processed data, often known as information or **results**.
- So far we have seen **two methods** of providing data to the program variables.
  1. One method is to assign values to variables through the assignment statements such as x= 5; a = 0; and so on.
  2. Another method is to use the input function **scanf** which can read data from a keyboard.
- For outputting results we have used extensively the function **printf** which sends results out to a terminal.
- All **input/output** operations are carried out through function calls such as **printf and scanf**.
- There exist several functions that have more or less become standard for input and output operations in C.
- These functions are collectively known as the **standard I/O library.**

READING A CHARACTER

- The simplest of all input/output operations is Reading & Writing a character.
- Reading a Character:
  - Can be done from the 'standard input' unit (usually the keyboard)
- Writing a Character:
  - writing it to the 'standard output' unit (usually the screen).

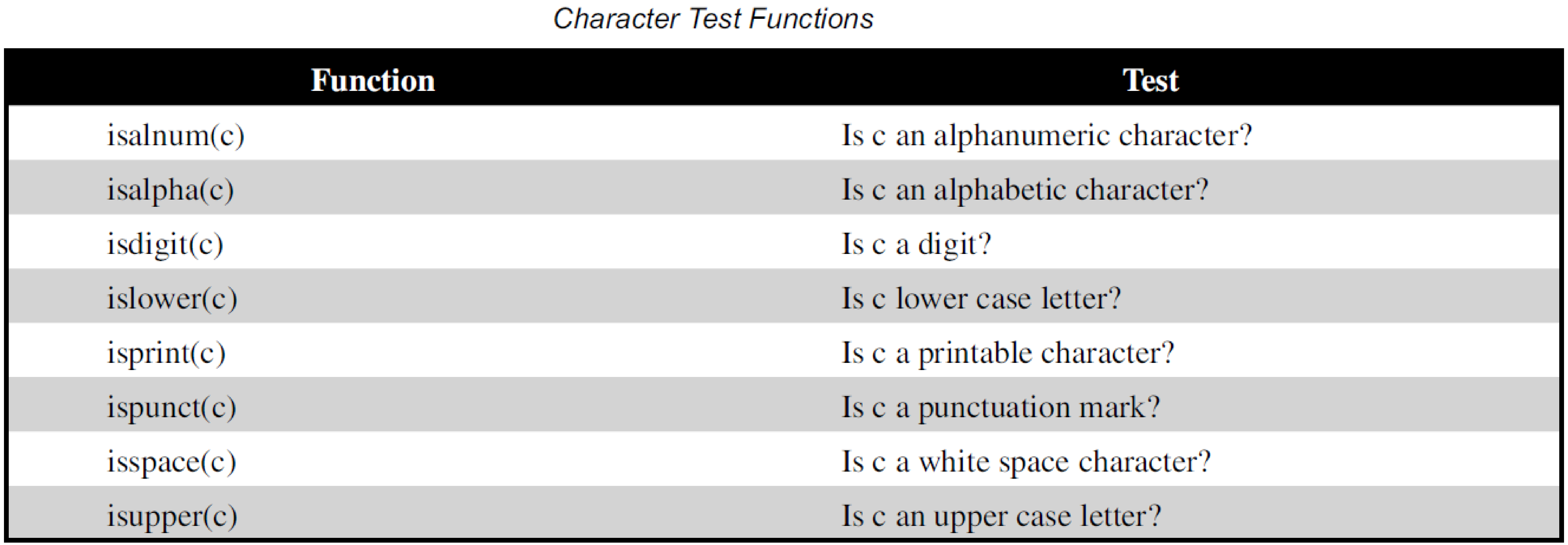
- Reading a single character can be done by using the function getchar. (This can also be done with the help of the scanf function)
- The getchar takes the following form:

```
getchar( );
variable_name = getchar( );
```

- variable_name is a valid C name that has been declared as **char** type.

READING A CHARACTER

- C supports many other similar functions as shown in below table.
- These character functions are contained in the file ctype.h and therefore the statement must be included in program as like:
- #include <ctype.h>



Character Test Functions

Function	Test
isalnum(c)	Is c an alphanumeric character?
isalpha(c)	Is c an alphabetic character?
isdigit(c)	Is c a digit?
islower(c)	Is c lower case letter?
isprint(c)	Is c a printable character?
ispunct(c)	Is c a punctuation mark?
isspace(c)	Is c a white space character?
isupper(c)	Is c an upper case letter?

WRITING A CHARACTER

- Like **getchar**, there is an analogous function **putchar** for writing characters one at a time to the terminal.
- It takes the form as shown below:

```
putchar (variable_name);
```

- where variable_name is a type **char** variable containing a character.
- This statement **displays the character** contained in the variable_name at the terminal.
- For example, the statements

```
answer = ‘Y’;
putchar (answer);
```

- The output will be displayed as character “Y” on the screen.

WRITING A CHARACTER

- The program uses **three** new functions
  - islower
  - toupper
  - tolower.

- islower:
- The function islower is a conditional function and takes the value TRUE if the argument is a lowercase alphabet; otherwise takes the value FALSE.

- toupper:
- The function toupper converts the **lowercase** argument into an **uppercase** alphabet.

- tolower:
- The function tolower converts the **uppercase** argument into a **lowercase** alphabet.

WRITING A CHARACTER

```
#include <stdio.h>
#include<conio.h>
#include <ctype.h>
void main()
{
        char alphabet;
        printf(“Enter an alphabet”);
        putchar(‘\n’); /* move to next line */
        alphabet = getchar();
        if (islower(alphabet))
                putchar(toupper(alphabet));/* Reverse and
                display */
        else
                putchar(tolower(alphabet)); /* Reverse and
                display */
}
```

Output

```
Enter an alphabet
a
A
Enter an alphabet
Q
q
Enter an alphabet
z
Z
```

UNFORMATTED INPUT OUTPUT STATEMENTS

- putchar( ) Function
- Single Character Output.
- Used to display one character at time on the standard output device.
- This function does the reverse operation of single character input i.e – getchar( ) function

- getc( ) Function
- used to accept a single character from the standard input to a character variable
- Ex: char c;
- c=getc( )

- putc( ) Function
- used to display a single character in a character variable to standard output device
- Ex: char a;
- putc(a)

UNFORMATTED INPUT OUTPUT STATEMENTS

- gets( ) Function
- Used to read the string (a group of characters) from the standard input device (Keyboard)
- Ex: gets(s)

- puts( ) Function
- Used to display the string (a group of characters) to the standard output device (screen)
- Ex: puts(s)

- getch( ) Function
- Reads a single character directly from the keyboard without echoing to the screen.
- Ex: getch( )

FORMATTED INPUT

- Formatted input refers to an **input data** that has been arranged in a particular format.
- For example, consider the following data:

```
15.75  123   John
```

- This line contains **three** pieces of data, arranged in a particular form.
  1. **First part -** of the data should be read into a variable **float**.
  2. **Second part -** into **int**
  3. **Third part -** into **char**.
- This is possible in C using the **scanf** function. (scanf means scan formatted.)
- General form of scanf :

```
scanf (“control string”, arg1, arg2, ...... argn);
```

- The **control string** specifies the field format in which the data is to be entered
- The **arguments** arg1, arg2, ..., argn specify the address of locations where the data is stored.
- Control string and arguments are separated by **commas**.
- Scanf statements must terminate (end) with semi colon **(;)**
- Ex:

```
Scanf(“%d”, &a);
```

INPUTTING INTEGER & REAL NUMBERS

- **Inputting Integer Numbers**
- General Form:

  scanf(“%d %d”, &num1, &num2);
- Example. if the input data typed is

  31426   50
  - %d – indicates **Int** type of **control string**
  - **& -** indicates the storage location of **Int**(address)
  - Scanf correctly assigns 31426 to num1 and 50 to num2.
- Assign format will be like : num1 = 31426 and num2 = 50
- **Inputting Real Numbers**
- General Form:

  scanf(“%f %f”, &num1, &num2);
- Example. if the input data typed is

  314.26   50
- Assign format will be like : num1 = 314.26 and num2 = 50.00

INPUTTING CHARACTER STRINGS

- General Form:

  scanf("%c %c", &word1, &word2);
- Example. if the input data typed is

  A B
- Assign format will be like : word1 = A and word2 = B

READING MIXED DATA TYPES

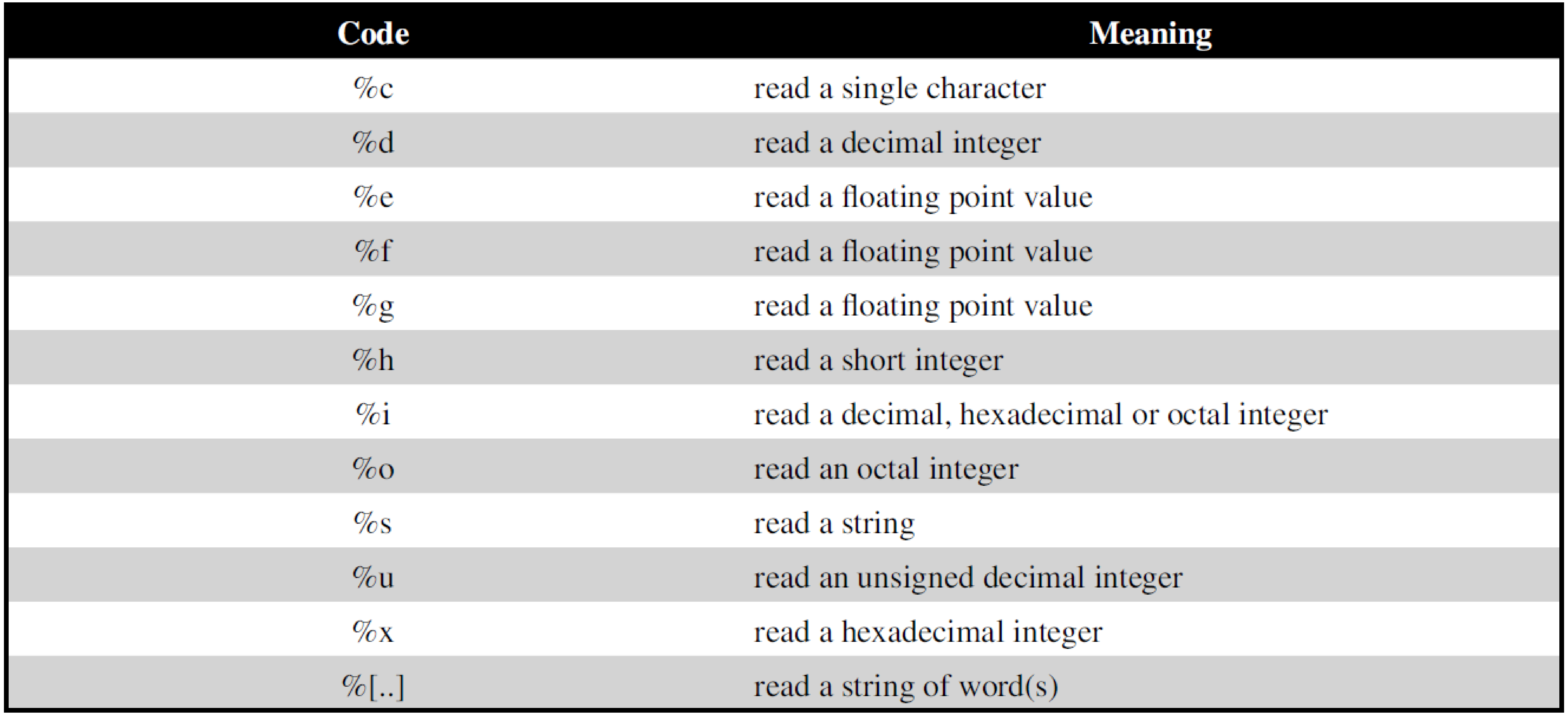
- General Form:

  scanf ("%d %c %f %s", &count, &code, &ratio, name);
- will read the data:

  15 p 1.575 coffee
- Will assign : count = 15, code = p, ratio = 1.575, name = coffee

FORMATTED INPUT

Commonly used scanf Format Codes



Code	Meaning
%c	read a single character
%d	read a decimal integer
%e	read a floating point value
%f	read a floating point value
%g	read a floating point value
%h	read a short integer
%i	read a decimal, hexadecimal or octal integer
%o	read an octal integer
%s	read a string
%u	read an unsigned decimal integer
%x	read a hexadecimal integer
%[..]	read a string of word(s)

Points to Remember while Using scanf

1. All function arguments, except the control string, must be pointers to variables.
2. Format specifications contained in the control string should match the arguments in order.
3. Input data items must be separated by spaces and must match the variables receiving the input in the same order.
4. The reading will be terminated, when scanf encounters a ‘mismatch’ of data or a character that is not valid for the value being read.
5. When searching for a value, scanf ignores line boundaries and simply looks for the next appropriate character.
6. Any unread data items in a line will be considered as part of the data input line to the next scanf call.

RULES FOR scanf

- Each variable to be read must have a field specification.
- For each field specification, there must be a variable address of proper type.
- Any non-whitespace character used in the format string must have a matching character in the user input.
- Never end the format string with whitespace. It is a fatal error!
- The scanf reads until:
    - A whitespace character is found in a numeric specification, or
    - The maximum number of characters have been read or
    - An error is detected, or
    - The end of file is reached

FORMATTED OUTPUT

- The printf statement provides certain features that can be effectively exploited to control the alignment and spacing of print-outs on the terminals.
- The general form of printf statement is:

  printf(“control string”, arg1, arg2, ..., argn);

- Control string consists of following **three** types of items:

  1. Characters that will be printed on the screen as they appear.
  2. Format specifications that define the output format for display of each item.
  3. Escape sequence characters such as \n, \t, and \b.

- Example: printf(“a = %f\n b = %f”, a, b);
- Input: if entered 10 20
- Output will be displayed as

  a = 10.00

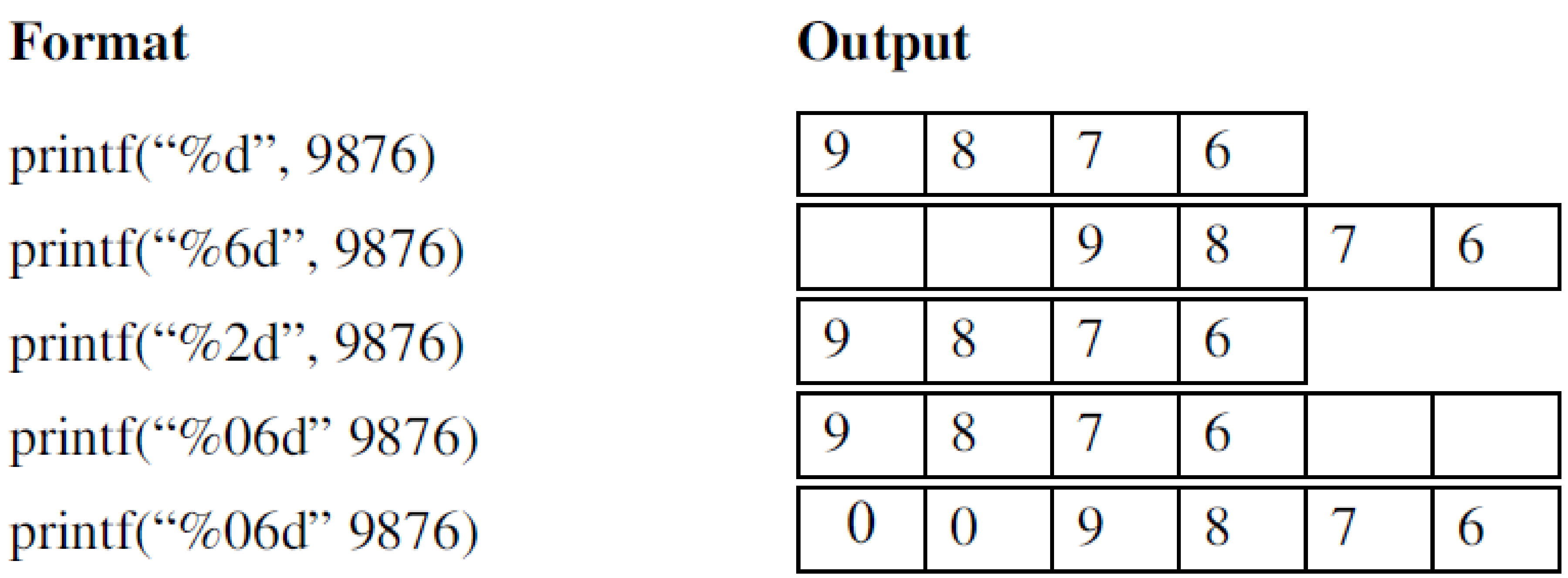
  b= 20.00

OUTPUT OF INTEGER NUMBERS

```
main()
{
        int m = 12345;
        long n = 987654;
        printf("%d\n",m);
        printf("%10d\n",m);
        printf("%010d\n",m);
        printf("%-10d\n",m);
        printf("%10ld\n",n);
        printf("%10ld\n",-n);
}
```

Output

```
12345
     12345
0000012345
12345
    987654
–   987654
```



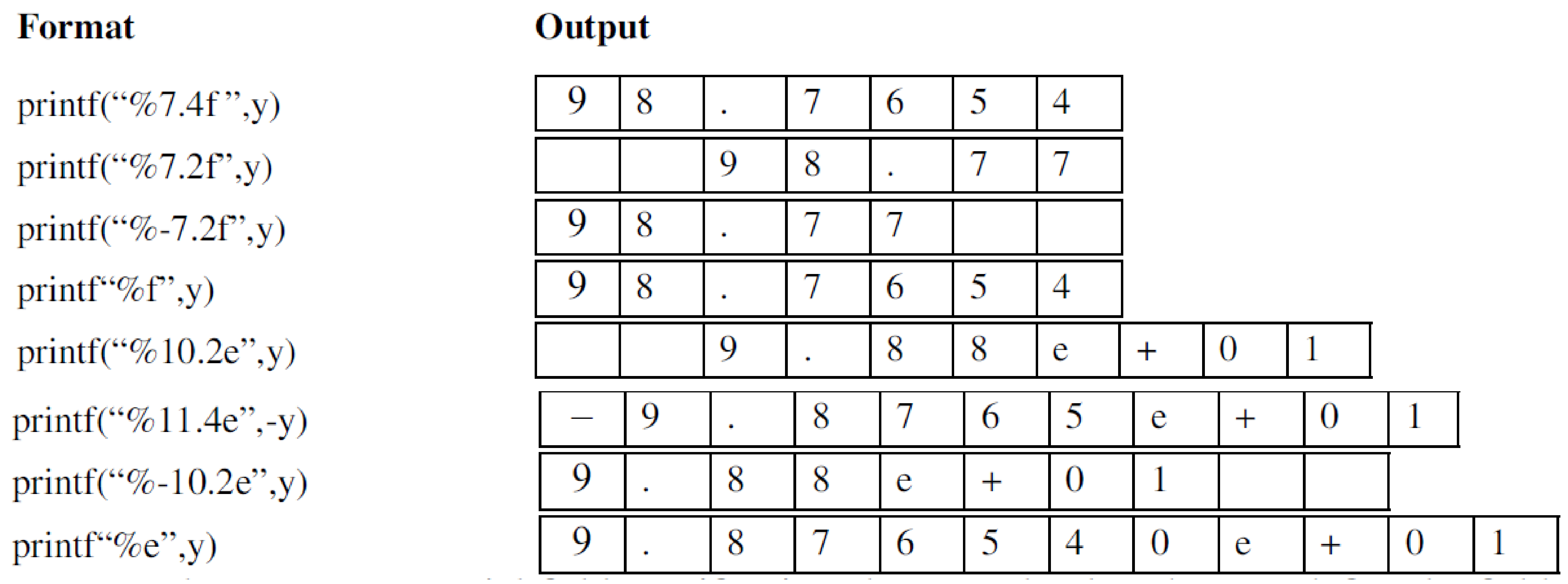
Format	Output					
printf("%d", 9876)	9	8	7	6		
printf("%6d", 9876)			9	8	7	6
printf("%2d", 9876)	9	8	7	6		
printf("%06d" 9876)	9	8	7	6		
printf("%06d" 9876)	0	0	9	8	7	6

Output of Real Numbers

- The output of a real number may be displayed in decimal notation.

```
main()
{
        float y = 98.7654;
        printf("%7.4f\n", y);
        printf("%f\n", y);
        printf("%7.2f\n", y);
        printf("%-7.2f\n", y);
        printf("%07.2f\n", y);
        printf("%*.*f", 7, 2, y);
        printf("\n");
        printf("%10.2e\n", y);
        printf("%12.4e\n", -y);
        printf("%-10.2e\n", y);
        printf("%e\n", y);
}
```

```
Output
98.7654
98.765404
98.77
98.77
0098.77
98.77
9.88e+001
-9.8765e+001
9.88e+001
9.876540e+001
```



Format	Output											
printf("%7.4f ",y)	9	8	.	7	6	5	4					
printf("%7.2f",y)			9	8	.	7	7					
printf("%-7.2f",y)	9	8	.	7	7							
printf"%f",y)	9	8	.	7	6	5	4					
printf("%10.2e",y)			9	.	8	8	e	+	0	1		
printf("%11.4e",-y)	–	9	.	8	7	6	5	e	+	0	1	
printf("%-10.2e",y)	9	.	8	8	e	+	0	1				
printf"%e",y)	9	.	8	7	6	5	4	0	e	+	0	1

MIXED DATA OUTPUT

- It is permitted to mix data types in one printf statement.
- For example, the statement of the type

  printf("%d %f %s %c", a, b, c, d);
- is valid.
- printf uses its control string to decide how many variables to be printed and what their types are.
- Therefore, the format specifications should match the variables in number, order, and type.
- If there are not enough variables or if they are of the wrong type, the output results will be incorrect.
- Enhancing the Readability of Output
- Correctness and clarity of outputs are of utmost importance.
- Correctness depends on the solution procedure
- Clarity depends on the way the output is presented.
- Following are some of the steps we can take to improve the clarity and hence the readability and understandability of outputs.
  1. Provide enough blank space between two numbers.
  2. Introduce appropriate headings and variable names in the output.
  3. Print special messages whenever a peculiar condition occurs in the output.
  4. Introduce blank lines between the important sections of the output.