HANDLING INPUT/OUTPUT

Unformatted and Formatted I/O

Designed by Parul Khurana, LIECA.

Standard I/O Functions

- There are many library functions available for standard I/O.
- These functions are divided into two categories:
  - **Unformatted**
  - **Formatted**

Standard I/O Functions

- The only difference between these functions is that the formatted functions permit the input from a standard input device or output sent to a standard output device to be **formatted** as per requirements.
  - For example, if *different values are to be displayed*, how much field width i.e., how many columns on screen, is to be used, and how much space between two values is to be given. If a value to be displayed is of real type, then how many decimal places to output.

Standard I/O Functions

- Unformatted I/O Functions
  - In the unformatted I/O category, we have functions for performing input/output of one character at a time, known as **character I/O functions** and functions that perform input/output of one string at a time, known as **string I/O functions**.

Standard I/O Functions

- Unformatted I/O Functions
  - **Character I/O functions**
    - **Single Character Input - THE getchar FUNCTION**
    - **Single Character Output - THE putchar FUNCTION**
  - **String I/O functions**
    - **Single String Input - THE gets FUNCTION**
    - **Single String Output - THE puts FUNCTION**

Standard I/O Functions

- Unformatted I/O Functions
  - **Character I/O functions**
    - **Single Character Input - THE getchar FUNCTION**
      - **Single characters can be entered into the computer using the C library function getchar.**
      - **The getchar function is a part of the standard C I/O library.**
      - **It returns a single character from a standard input device (typically a keyboard).**
      - **The function does not require any arguments, though a pair of empty parentheses must follow the word getchar.**

Standard I/O Functions

- Unformatted I/O Functions
  - **Character I/O functions**
    - **Single Character Input - THE getchar FUNCTION**
      - **In general terms, a function reference would be written as:**
        - » ***character variable = getchar();***

          where *character variable* refers to some previously declared character variable.

Standard I/O Functions

- Unformatted I/O Functions
  - **Character I/O functions**
    - **Single Character Input - THE getchar FUNCTION - EXAMPLE**
      - **A C program contains the following statements:-**

```
char c;
……..
c = getchar();
```

      - **The first statement declares that c is a character-type variable.**
      - **The second statement causes a single character to be entered from the standard input device (usually a keyboard) and then assigned to c.**

Standard I/O Functions

- Unformatted I/O Functions
  - **Character I/O functions**
    - **Single Character Output - THE putchar FUNCTION**
      - **Single characters can be displayed (i.e., written out of the computer) using the C library function putchar.**
      - **The putchar function, like getchar function is a part of the standard C I/O library.**
      - **It transmits a single character to a standard output device (typically a monitor). The character being transmitted will normally be represented as a character-type variable.**
      - **It must be expressed as an argument to the function, enclosed in parentheses, following the word putchar.**

Standard I/O Functions

- Unformatted I/O Functions
  - **Character I/O functions**
    - **Single Character Output - THE putchar FUNCTION**
      - **In general , a function reference would be written as:**
        - » ***putchar(character variable);***

          where *character variable* refers to some previously declared character variable.

Standard I/O Functions

- Unformatted I/O Functions
  - **Character I/O functions**
    - **Single Character Output - THE putchar FUNCTION - EXAMPLE**
      - **A C program contains the following statements:-**

```
char c;
……..
putchar(c);
```

      - **The first statement declares that c is a character-type variable.**
      - **The second statement causes the current value of c to be transmitted to the standard output device (usually a monitor) where it will be displayed.**

Standard I/O Functions

- Unformatted I/O Functions
  - **String I/O functions**
    - **Single String Input - THE gets FUNCTION**
      - **Multiple characters can be entered into the computer using the C library function gets.**
      - **The gets function is a part of the standard C I/O library and it offers alternative to the use of scanf function.**
      - **The string will be entered from the keyboard, and will terminate with a newline character (i.e., the string will end when the user presses the Enter key).**
      - **The function accepts a single argument. The argument must be a data item that represents a string, (e.g., a character array).**

Standard I/O Functions

- Unformatted I/O Functions
  - **String I/O functions**
    - **Single String Input - THE gets FUNCTION**
      - **In general terms, a function reference would be written as:**
        - » ***gets(character array);***

          where *character array* refers to some previously declared character array with length/size specified (e.g., char line[80]).

Standard I/O Functions

- Unformatted I/O Functions
  - **String I/O functions**
    - **Single String Input - THE gets FUNCTION - EXAMPLE**
      - **A C program contains the following statements:-**

```
char line[80];
……..
gets(line);
```

      - **The first statement declares that line is a character-type array of size 80.**
      - **The second statement causes a multiple characters to be entered from the standard input device (usually a keyboard) and then assigned to line.**

Standard I/O Functions

- Unformatted I/O Functions
  - **String I/O functions**
    - **Multiple Characters Output - THE puts FUNCTION**
      - **Multiple characters can be displayed (i.e., written out of the computer) using the C library function puts.**
      - **The puts function, like gets function is a part of the standard C I/O library.**
      - **It transmits a character array to a standard output device (typically a monitor). The character array being transmitted will normally be represented as a character-type array.**
      - **It must be expressed as an argument to the function, enclosed in parentheses, following the word puts.**

Standard I/O Functions

- Unformatted I/O Functions
  - **String I/O functions**
    - **Single String Output - THE puts FUNCTION**
      - **In general terms, a function reference would be written as:**
        - » ***puts(character array);***

          where *character array* refers to some previously declared character array with length/size specified (e.g., char line[80]).

Standard I/O Functions

- Unformatted I/O Functions
  - **String I/O functions**
    - **Single String Output - THE puts FUNCTION - EXAMPLE**
      - **A C program contains the following statements:-**

```
char line[80];
……..
puts(line);
```

      - **The first statement declares that line is a character-type array of size 80.**
      - **The second statement causes the current value of line to be transmitted to the standard output device (usually a monitor) where it will be displayed.**

Standard I/O Functions

- Formatted I/O Functions
  - Entering INPUT DATA - THE scanf FUNCTION
  - MORE ABOUT THE scanf FUNCTION
  - Writing OUTPUT DATA - THE printf FUNCTION
  - MORE ABOUT THE printf FUNCTION

Standard I/O Functions

- Formatted I/O Functions
  - Entering INPUT DATA - THE scanf FUNCTION
    - Input data can be entered into the computer from a standard input device by means of the C library function **scanf**.
    - This function can be used to enter any combination of numerical values, single characters and strings.
    - The ***function returns the number of data items*** that have been entered successfully.

Standard I/O Functions

- Formatted I/O Functions
  - In general terms, the **scanf function** is written as:
    - ***scanf (control string, arg1, arg2, ……., argN);***
      - **where *control string* refers to a string containing certain required formatting information, and arg1, arg2, ……., argN are arguments that represent the individual input data items.**
      - **The control string consists of individual groups of characters, with one character group for each input data item. Each character group must begin with a percent sign (%).**
      - **In its simplest form, a single character group will consist of the percent sign, followed by a conversion character which indicates the type of the corresponding data item.**

Standard I/O Functions

- Formatted I/O Functions
  - In general terms, the **scanf function** is written as:
    - ***scanf (control string, arg1, arg2, ……., argN);***
      - **Within the control string, multiple character groups can be contiguous, or they can be separated by whitespace characters (i.e., *blank spaces*, *tabs* or *newline characters*).**
      - **If whitespace characters are used to separate multiple character groups in the control string, then all consecutive whitespace characters in the input data will be read but ignored.**
      - ***The use of blank spaces as character-group separators is very common.***

Standard I/O Functions

- Formatted I/O Functions
  - In general terms, the **scanf function** is written as:
    - ***scanf (control string, arg1, arg2, ……., argN);***
      - **The more frequently used conversion characters are listed below:-**

Conversion Character	Meaning
c	data item is a single character.
d	data item is a signed decimal integer.
e	data item is a floating-point value(exponential notation).
f	data item is a single precision floating-point value.
h	data item is a short integer.
g	data item is a floating-point value without leading or trailing zeroes.
o	data item is an octal integer.

Standard I/O Functions

- Formatted I/O Functions
  - In general terms, the **scanf function** is written as:
    - ***scanf (control string, arg1, arg2, ……., argN);***
      - **The more frequently used conversion characters are listed below:-**

Conversion Character	Meaning
s	data item is a **single word** string followed by a whitespace character (the null character \0 will automatically be added at the end).
[^\n]s	data item is a **multi word** string followed by a whitespace character (the null character \0 will automatically be added at the end).
u	data item is an unsigned decimal integer.
x	data item is a hexadecimal integer.

Standard I/O Functions

- Formatted I/O Functions
  - In general terms, the **scanf function** is written as:
    - ***scanf (control string, arg1, arg2, ……., argN);***
      - **The more frequently used conversion characters are listed below:-**

Conversion Character	Meaning
ld	data item is signed long decimal integer.
lf	data item is double precision floating-point value.
[…]	data item is a string which may include whitespace characters. *< The arguments are written as variables or arrays, whose types match the corresponding character groups in control string.>*

Standard I/O Functions

- Formatted I/O Functions
  - Entering INPUT DATA - THE scanf FUNCTION - TYPICAL APPLICATION -1

```
#include <stdio.h>
void main(void)
 {
   char item[20];
    int partno;
    float cost;
    . . . . . .
    scanf ( " %s  %d  %f  " ,  item, &partno,  &cost);
    . . . . . .
 }
```

✓**Within the scanf function, the control string is "%s %d %f".**

✓**It contains three character groups.**

✓**The first character group, %d, indicates that the second argument (&partno) represents a *decimal integer value*, and the third character group, %f, indicates that the third argument (&cost) represents a floating-point value.**

Standard I/O Functions

- Formatted I/O Functions
  - Entering INPUT DATA - THE scanf FUNCTION - TYPICAL APPLICATION -1

```
#include <stdio.h>
void main(void)
 {
   char item[20];
    int partno;
    float cost;
    . . . . . .
    scanf ( " %s  %d  %f  " ,  item, &partno,  &cost);
    . . . . . .
 }
```

✓**Notice that the numerical variables partno and cost are preceded by ampersands within the scanf function.**

✓**An ampersand does not precede item, however since item is an array name and an implicit memory address reference.**

Standard I/O Functions

- Formatted I/O Functions
  - Entering INPUT DATA - THE scanf FUNCTION - TYPICAL APPLICATION -2

```
#include <stdio.h>
void main(void)
 {
   char line[80];
    . . . . . .
   scanf ( " %[ ABCDEFGHIJKLMNOPQRSTUVWXYZ]" , line);
    . . . . . .
 }
```

- ✓This example illustrates the use of the scanf function to enter a string consisting of uppercase letters and blank spaces.
- ✓The string will be of undetermined length, but it will be limited to 79 characters (actually, 80 characters including the null character that is added at the end).
- ✓Notice the blank space that precedes the % sign.

Standard I/O Functions

- Formatted I/O Functions
  - Entering INPUT DATA - THE scanf FUNCTION - TYPICAL APPLICATION -2

```
#include <stdio.h>
void main(void)
 {
   char line[80];
    . . . . . .
   scanf ( " %[ ABCDEFGHIJKLMNOPQRSTUVWXYZ]" , line);
    . . . . . .
 }
```

✓If the string

✓NEW YORK CITY

✓Is entered from the standard input device when the program is executed, the entire string will be assigned to the array line since the string is comprised entirely of uppercase letters and blank spaces.

Standard I/O Functions

- Formatted I/O Functions
  - Entering INPUT DATA - THE scanf FUNCTION - TYPICAL APPLICATION -2

```
#include <stdio.h>
void main(void)
  {
     char line[80];
      . . . . . .
     scanf ( " %[ ABCDEFGHIJKLMNOPQRSTUVWXYZ]" , line);
      . . . . . .
   }
```

- ✓If the string were written as
- ✓New York City
- ✓However, then only the single letter N would be assigned to line, since first lowercase letter (in this case, e) would be interpreted as the *first character beyond the string*.

Standard I/O Functions

- Formatted I/O Functions
  - MORE ABOUT THE scanf FUNCTION
    - The consecutive non-whitespace characters that define a data item collectively define a field.
    - It is possible to limit the number of such characters by specifying a maximum field width for that data item.
      - ***To do so, an unsigned integer indicating the field width is placed within the control string, between (%) sign and the conversion character.***

Standard I/O Functions

- Formatted I/O Functions
  - MORE ABOUT THE scanf FUNCTION
    - **The data item may contain fewer characters than the specified field width.**
    - ***However, the number of characters in the actual data item cannot exceed the specified field width.***
    - ***Any characters that extend beyond the specified field width will not be read.***
    - ***Such leftover characters may be incorrectly interpreted as the components of the next data item.***

Standard I/O Functions

- Formatted I/O Functions

  – MORE ABOUT THE scanf FUNCTION- EXAMPLE -1

```
#include <stdio.h>
void main(void)
 {
    int a, b, c;
     . . . . . .
    scanf ( " %3d  %3d  %3d " , &a,  &b,  &c);
     . . . . . .
 }
```

✓When the program is executed, three integer quantities will be entered from the standard input device (the keyboard)

Standard I/O Functions

- Formatted I/O Functions

  – MORE ABOUT THE scanf FUNCTION- EXAMPLE -1

```
#include <stdio.h>
void main(void)
 {
   int a, b, c;
    . . . . . .
   scanf ( " %3d  %3d  %3d " , &a,  &b,  &c);
    . . . . . .
 }
```

✓**Scenario -1**

✓**Suppose the input data items are entered as:**

✓**1 2 3**

✓**Then the following assignments will result:**

a = 1, b = 2, c = 3

Standard I/O Functions

- Formatted I/O Functions

  – MORE ABOUT THE scanf FUNCTION- EXAMPLE -1

```
#include <stdio.h>
void main(void)
 {
   int a, b, c;
    . . . . . .
   scanf ( " %3d  %3d  %3d " , &a,  &b,  &c);
    . . . . . .
 }
```

✓**Scenario -2**

✓**If the data had been entered as:**

✓**123 456 789**

✓**Then the assignments would be:**

a = 123, b = 456, c = 789

Standard I/O Functions

- Formatted I/O Functions
  - MORE ABOUT THE scanf FUNCTION- EXAMPLE -1

```
#include <stdio.h>
void main(void)
  {
     int a, b, c;
      . . . . . .
     scanf ( " %3d   %3d   %3d " , &a,   &b,   &c);
      . . . . . .
   }
```

✓**Scenario -3**

✓**Now suppose the data had been entered as:**

✓**123456789**

✓**Then the assignments would be:**

a = 123, b = 456, c = 789

  - **As before, since the first three digits would be assigned to a, the next three digits to b, and the last three digits to c.**

Standard I/O Functions

- Formatted I/O Functions
  - MORE ABOUT THE scanf FUNCTION- EXAMPLE -1

```
#include <stdio.h>
void main(void)
  {
     int a, b, c;
      . . . . . .
     scanf ( " %3d  %3d  %3d " , &a,  &b,  &c);
      . . . . . .
  }
```

✓**Scenario -4**

✓**Finally, suppose that the data had been entered as:**

✓**1234 5678 9**

✓**The resulting assignments would now be:**

a = 123, b = 4, c = 567

  - **The remaining two digits (8 and 9) would be ignored, unless they were read by a subsequent statement.**

Standard I/O Functions

- Formatted I/O Functions
  - Writing OUTPUT DATA - THE printf FUNCTION
    - Output data can be written from the computer onto a standard output device using the library function **printf**.
    - This function can be used to output any combination of numerical values, single characters and strings.
    - This ***function moves data from the computer's memory to the standard output device, whereas the scanf function enters data from the standard input device and stores it in the computer's memory.***

Standard I/O Functions

- Formatted I/O Functions
  - In general terms, the **printf function** is written as:
    - ***printf (control string, arg1, arg2, ……., argN);***
      - **where *control string* refers to a string that contains formatting information, and arg1, arg2, ……., argN are arguments that represent the individual output data items.**
      - **The arguments can be written as constants, single variable or array names, or more complex expressions. Function references may also be included.**
      - ***In contrast to the scanf function discussed earlier, the arguments in a printf function do not represent memory address and therefore are not preceded by ampersands.***

Standard I/O Functions

- Formatted I/O Functions
  - In general terms, the **printf function** is written as:
    - ***printf (control string, arg1, arg2, ……., argN);***
      - **The control string consists of individual groups of characters, with one character group for each output data item. Each character group must begin with a percent sign (%).**
      - **In its simplest form, an individual character group will consist of the percent sign, followed by a conversion character indicating the type of the corresponding data item.**

Standard I/O Functions

- Formatted I/O Functions
  - In general terms, the **printf function** is written as:
    - ***printf (control string, arg1, arg2, ……., argN);***
      - **Multiple character groups can be contiguous, or they can be separated by other characters, including whitespace characters.**
      - **These "other" characters are simply transferred directly to the output device where they are displayed.**
      - **The use of blank spaces as character-group separators is particularly common.**

Standard I/O Functions

- Formatted I/O Functions
  - Writing OUTPUT DATA - THE printf FUNCTION - TYPICAL APPLICATION -1

```
#include <stdio.h>
void main(void)
 {
    char item[20];
     int partno;
     float cost;
      . . . . . .
     printf ( " %s  %d  %f  " , item, partno,  cost);
      . . . . . .
 }
```

✓Within the printf function, the control string is "%s %d %f".

✓It contains three character groups.

✓The first character group, %s, indicates that the first argument (item) represents a ***string***.

✓ The second character group, %d, indicates that the second argument (partno) represents a decimal integer value .

Standard I/O Functions

- Formatted I/O Functions
  - Writing OUTPUT DATA - THE printf FUNCTION - TYPICAL APPLICATION -1

```
#include <stdio.h>
void main(void)
 {
    char item[20];
     int partno;
     float cost;
      . . . . . .
     printf ( " %s  %d  %f  " ,  item, partno,  cost);
      . . . . . .
 }
```

✓Within the printf function, the control string is "%s %d %f".

✓It contains three character groups.

✓The third character group, %f, indicates that the third argument (cost) represents a floating-point value.

Standard I/O Functions

- Formatted I/O Functions
  - Writing OUTPUT DATA - THE printf FUNCTION - TYPICAL APPLICATION -1

```
#include <stdio.h>
void main(void)
 {
   char item[20];
    int partno;
    float cost;
    . . . . . .
    printf ( " %s  %d  %f  " ,  item, partno,  cost);
    . . . . . .
 }
```

✓**Scenario 1**

✓**Now suppose that name, partno and cost have been assigned the values fastener, 12345 and 0.05, respectively, within the program.**

✓**When the printf statement is executed, the following output will be generated.**

```
fastener  12345  0.050000
```

Standard I/O Functions

- Formatted I/O Functions
  - Writing OUTPUT DATA - THE printf FUNCTION - TYPICAL APPLICATION -1

```
#include <stdio.h>
void main(void)
 {
    char item[20];
     int partno;
     float cost;
      . . . . . .
     printf ( "%s%d%f" ,  item, partno,  cost);
      . . . . . .
 }
```

✓**Scenario 2**

✓**Now suppose that the printf statement had been written as :**

✓ **This printf statement is syntactically valid, though it causes the output to run together; i.e.,**

```
fastener123450.050000
```

Standard I/O Functions

- Formatted I/O Functions
  - Writing OUTPUT DATA - THE printf FUNCTION
    - **A minimum field width can be specified by preceding the conversion character by an unsigned integer.**
    - ***If the number of characters in the corresponding data item is less than the specified field width, then the data item will be preceded by enough leading blanks to fill the specified field.***
    - ***If the number of characters in the data item exceeds the specified field width, however, then additional space will be allocated to the data item, so that entire data item will be displayed.***

Standard I/O Functions

- Formatted I/O Functions
  - Writing OUTPUT DATA - THE printf FUNCTION
    - ***This is just the opposite of the field width indicator in the scanf function, which specifies a maximum field width.***

Standard I/O Functions

- Formatted I/O Functions
  - Writing OUTPUT DATA - THE printf FUNCTION - TYPICAL APPLICATION -2

```
#include <stdio.h>
void main(void)
  {
     int i = 12345;
     float x = 345.678;
      printf ( "%3d  %5d  %8d\n\n" ,  i, i, i);
      printf ( "%3f  %10f  %13f\n\n" ,  x, x, x);
      printf ( "%3e  %13e  %16e" ,  x, x, x);
  }
```

✓**Notice the double newline character in the first two printf statements.**

✓**They will cause the lines of output to be doubled spaced, as shown below.**

```
12345  12345    12345

345.678000  345.678000    345.678000

3.456780e+02  3.456780e+02    3.456780e+02
```

Standard I/O Functions

- Formatted I/O Functions
  - Writing OUTPUT DATA - THE printf FUNCTION - TYPICAL APPLICATION -2

```
#include <stdio.h>
void main(void)
 {
   int i = 12345;
   float x = 345.678;
    printf ( "%3d  %5d  %8d\n\n" ,  i, i, i);
    printf ( "%3f  %10f  %13f\n\n" ,  x, x, x);
    printf ( "%3e  %13e  %16e" ,  x, x, x);
 }
```

```
12345  12345    12345

345.678000  345.678000    345.678000

3.456780e+02  3.456780e+02    3.456780e+02
```

✓**The first line of output displays a decimal integer using three different minimum field widths (three, five and eight characters).**

✓**The entire integer value is displayed within each field, even if the field width is too small (as with the first field in this example).**

Standard I/O Functions

- Formatted I/O Functions
  - Writing OUTPUT DATA - THE printf FUNCTION - TYPICAL APPLICATION -2

```
#include <stdio.h>
void main(void)
 {
   int i = 12345;
   float x = 345.678;
    printf ( "%3d  %5d  %8d\n\n" ,  i, i, i);
    printf ( "%3f  %10f  %13f\n\n" ,  x, x, x);
    printf ( "%3e  %13e  %16e" ,  x, x, x);
 }
```

```
12345  12345     12345

345.678000  345.678000     345.678000

3.456780e+02  3.456780e+02     3.456780e+02
```

✓**The second value in the first line is preceded by one blank space.**

✓**This is generated by the blank space separating the first two character groups within the control string.**

Standard I/O Functions

- Formatted I/O Functions
  - Writing OUTPUT DATA - THE printf FUNCTION - TYPICAL APPLICATION -2

```
#include <stdio.h>
void main(void)
 {
   int i = 12345;
   float x = 345.678;
   printf ( "%3d  %5d  %8d\n\n" ,  i, i, i);
   printf ( "%3f  %10f  %13f\n\n" ,  x, x, x);
   printf ( "%3e  %13e  %16e" ,  x, x, x);
 }
```

```
12345  12345    12345

345.678000  345.678000    345.678000

3.456780e+02  3.456780e+02    3.456780e+02
```

✓**The third value is preceded by four blank spaces.**

✓**One blank space comes from the blank space separating the last two character groups within the control field.**

✓**The other three blank spaces fill the minimum field width, which exceeds the number of characters in the output value.**

Standard I/O Functions

- Formatted I/O Functions
  - MORE ABOUT THE printf FUNCTION
    - *As we have already learned how to specify a minimum field width in a printf function.*
    - It is also possible to specify the maximum number of decimal places for a floating-point value, or the maximum number of characters for a string.
      - This specification is known as precision.
        - » The precision is an unsigned integer that is always preceded by a decimal point.
        - » If a minimum field width is specified in addition to the precision (as is usually the case), then the precision specification follows the field width specification.
        - » *Both of these integer specifications* precede the conversion character.

Standard I/O Functions

- Formatted I/O Functions
  - MORE ABOUT THE printf FUNCTION - TYPICAL APPLICATION -1

```
#include <stdio.h>
void main(void)
  {
   float x = 123.456;
   printf ( "%7f  %7.3f  %7.1f\n\n" ,  x, x, x);
   printf ( "%12e  %12.5e  %12.3e" ,  x, x, x);
   }
```

✓**When this program is executed, the following output is generated.**

```
123.456000  123.456      123.5

1.234560e+02   1.23456e+02      1.235e+02
```

Standard I/O Functions

- Formatted I/O Functions
  - MORE ABOUT THE printf FUNCTION - TYPICAL APPLICATION -1

```
#include <stdio.h>
void main(void)
  {
   float x = 123.456;
   printf ( "%7f  %7.3f  %7.1f\n\n" ,  x, x, x);
   printf ( "%12e  %12.5e  %12.3e" ,  x, x, x);
   }
```

```
123.456000  123.456      123.5

1.234560e+02   1.23456e+02      1.235e+02
```

✓**The first line is produced by f-type conversion.**

✓**Notice the rounding that occurs in the third number because of precision (one decimal place).**

✓**Also, notice the leading blanks that are added to fill the specified minimum field width (seven characters).**

Standard I/O Functions

- Formatted I/O Functions
  - MORE ABOUT THE printf FUNCTION - TYPICAL APPLICATION -1

```
#include <stdio.h>
void main(void)
  {
   float x = 123.456;
   printf ( "%7f  %7.3f  %7.1f\n\n" ,  x, x, x);
   printf ( "%12e  %12.5e  %12.3e" ,  x, x, x);
   }
```

```
123.456000  123.456      123.5

1.234560e+02   1.23456e+02      1.235e+02
```

✓The second line, produced by e-type conversion, has similar characteristics.

✓Again, we see that the third number is rounded to conform to the specified precision (three decimal places).

✓Also, note the leading blanks that are added to fill the specified minimum field width (twelve characters).

Practice Questions

- Give the output of the following code segment:

```
int i = -3, j = 2, k = 0, m;

m = ++i && j++ && ++k;

printf ( “%d %d %d %d”, i, j ,k , m);

i = -3, j = 2, k = 0, m;

m = ++i || j++ || ++k;

printf ( “\n%d %d %d %d”, i, j ,k , m);

printf ( “\n%.3f”, 12.345678);
```

Practice Questions

- Give the output of the following code segment:

```
int x = 10, y;
y = x++ + ++x + x++ + ++x + x++;
printf ( “\n %d %d”, x, y);
x = 10;
x = x++ + ++x + x++ + ++x + x++;
printf ( “\n %d %o %x”, x, x, x);
```