C_24 ⇒ Formatted Output Function in C

printf()

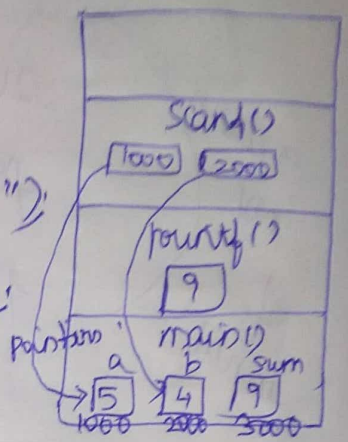
* stdio.h
  ↳ Standard input/output.
* Standard input means, when we can take input from user only through keyboard but not thro' printer or any other
* standard output means, we can see output in only monitor.

* printf ("control string", arg1, arg2 .... argn);

* printf ("only Message");

printf ("Hello world");

```
int a, b, sum;
printf ("Enter 2 numbers: ");
scanf ("%d %d", &a &b);
sum = a+b;
printf ("%d", sum);
// printf ("sum = %d", sum);
// printf ("a = %d, b = %d, sum = %d", a, b, sum);
// printf ("a = %d \n b = %d \n sum = %d \n", a, b, sum);
```



9

sum = 9

a = 5 b = 4 sum = 9

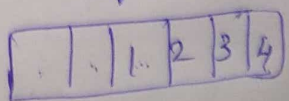
newline

a = 5
b = 4
sum = 9

Note:

```
a = 1234
printf ("%6d", a);
```

6 spaces

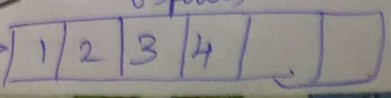


Right justified

Left justified

```
printf ("-6d", a);
```

6 spaces



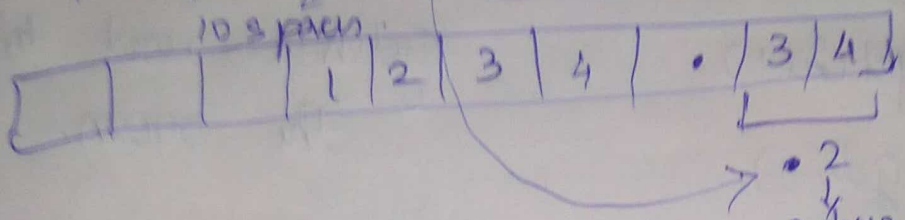
printf("%3d", a); ⇒ | 1 | 2 | 3 | 4 |

printf("%0d", a); ⇒ | d0 | 1 | 2 | 3 | 4 |

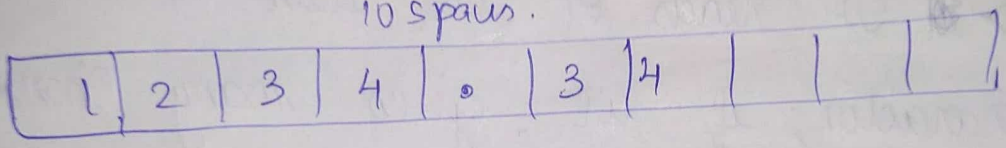
printf("%-0d", a) ⇒ | 1 | 2 | 3 | 4 | Hdvk |

a = 1234.3456

printf("%10.2f", a); ⇒ Right justified.



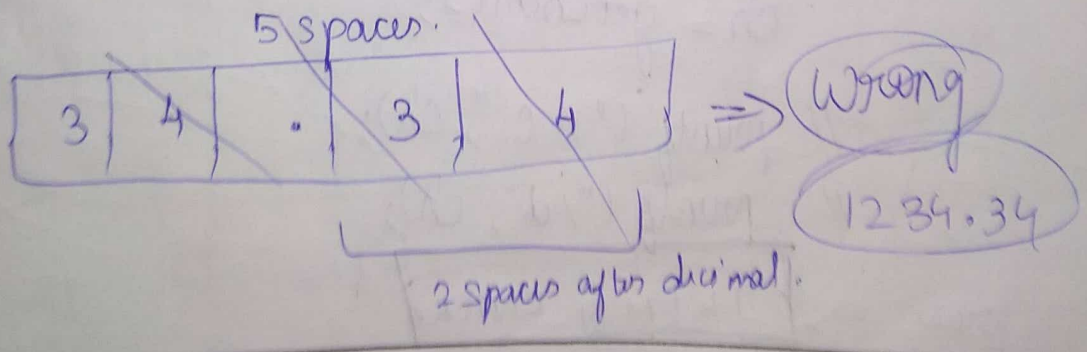
printf("%-10.2f", a); ⇒ Left justified



printf("%f", a); ⇒ 1234.345600

⇒ after decimal point 6 digits precisions.

printf("5.2%f", a); a = 1234.3456.



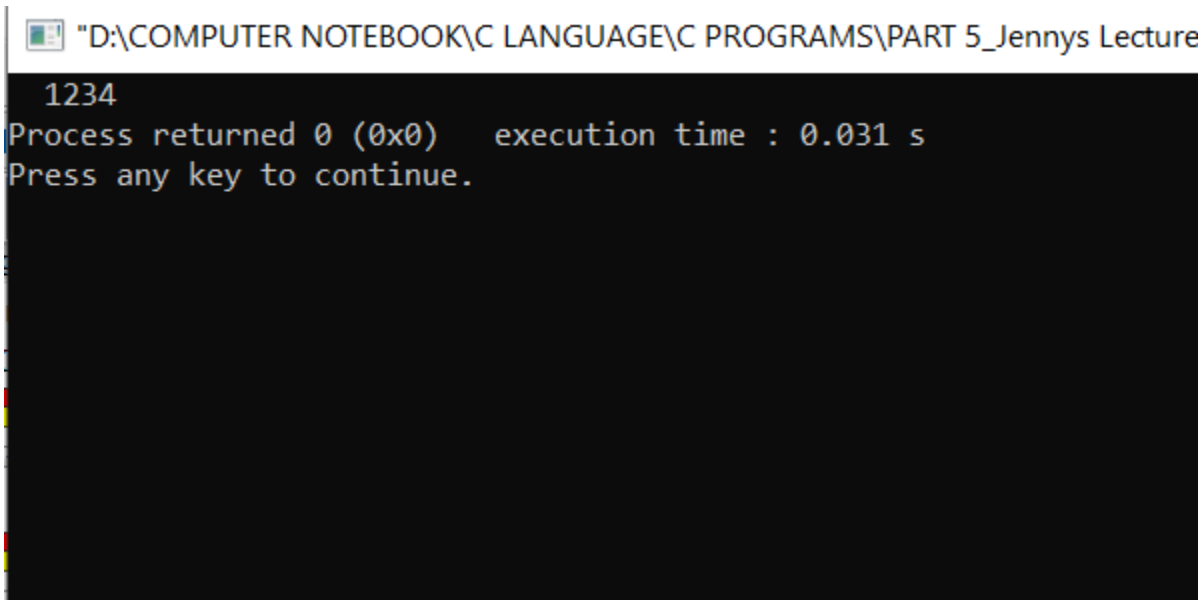
printf("%e", a);

a = 1234.3456;

$e^{-1} \Rightarrow \frac{1}{10}$.

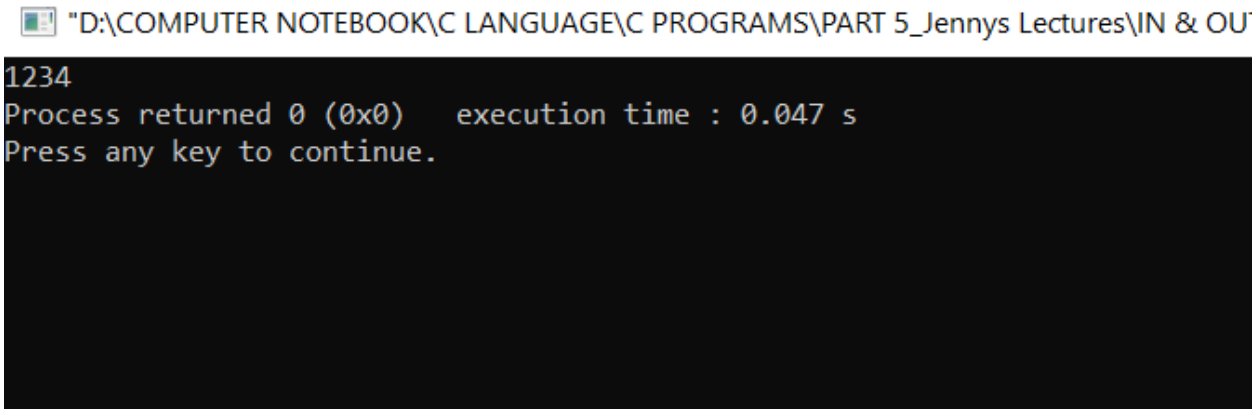
CODE 1:

```c
#include <stdio.h>
#include <stdlib.h>
/* 1 FORMATTED OUTPUT FUNCTIONS */
int main()
{
    int a=1234;
    printf("%6d",a);
 /* 6 spaces of memory are assigned & printed right justified by
 leaving first two spaces empty since we give only 4 digits */
}

```



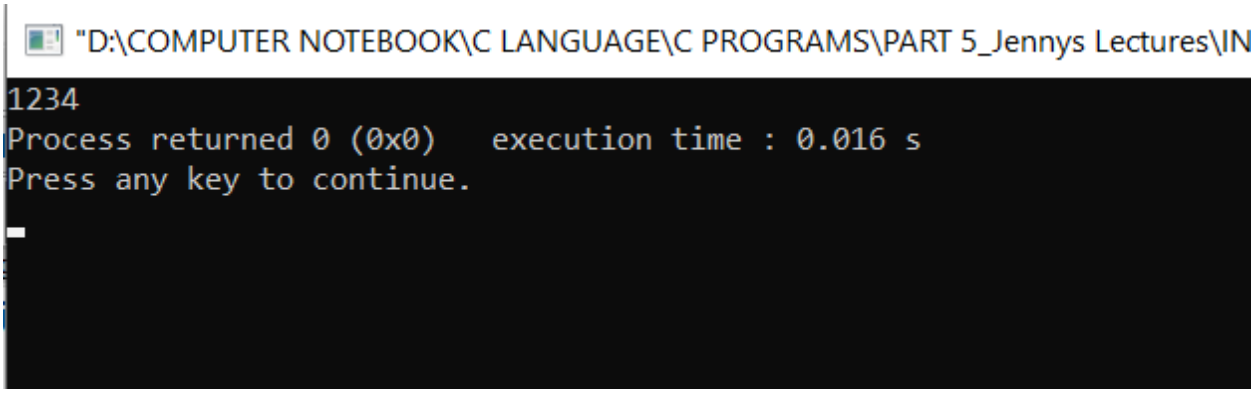
CODE 2:

```c
#include <stdio.h>
#include <stdlib.h>
/* 2 FORMATTED OUTPUT FUNCTIONS */
int main()
{
        int a=1234;
    printf("%-6d",a);
 /* 6 spaces of memory are assigned & printed left justified by
 leaving last two spaces empty since we give only 4 digits */
}

```



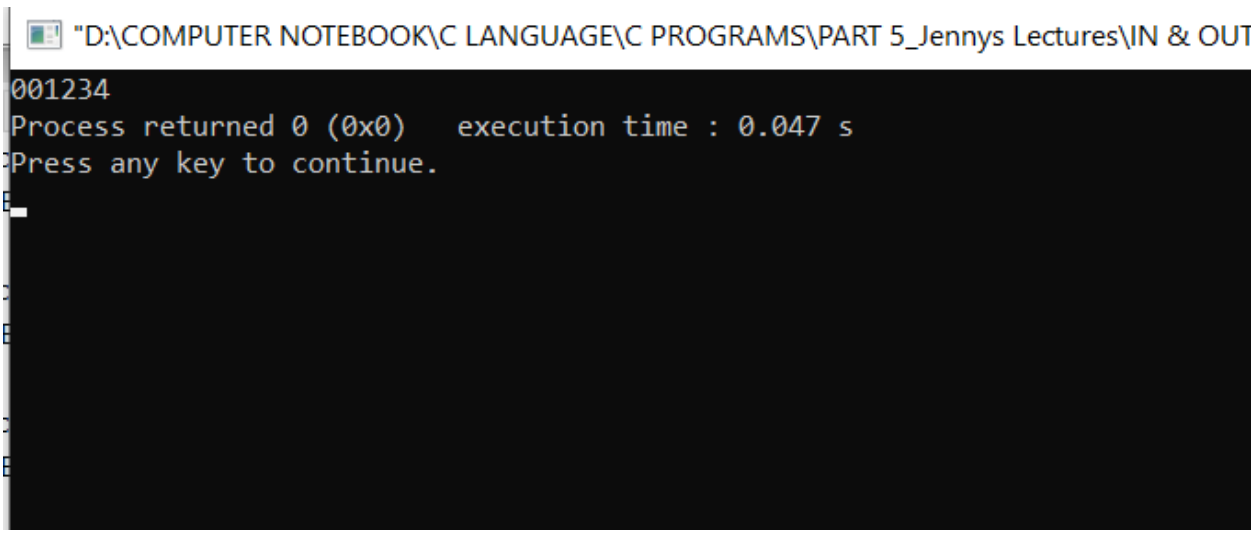
CODE 3:

```c
#include <stdio.h>
#include <stdlib.h>
/* 3 FORMATTED OUTPUT FUNCTIONS */
int main()
{
    int a=1234;
    printf("%3d",a);
 /* Even though we give 4 digits and 3 spaces of memory are
 assigned there is no limitations for formatted output
 functions .
 It will print the output as it...
 It will only align or justify the output*/
}
```



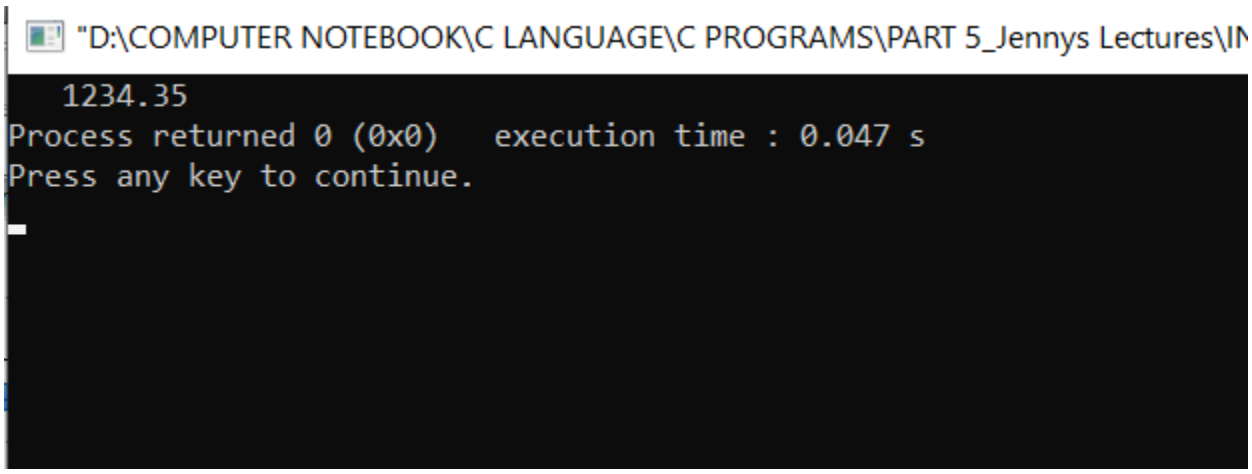
CODE 4:

```c
#include <stdio.h>
#include <stdlib.h>
/* 4 FORMATTED OUTPUT FUNCTIONS */
int main()
{
 int a=1234;
    printf("%06d",a);
 /* 6 spaces of memory are assigned & printed right justified
 by leaving first two spaces filled with 0 since we give
 only 4 digits */
 //printf("%-06d",a);
 /*6 spaces of memory are assigned & printed left justified
 by leaving last two spaces empty but not with 0....
 }
```



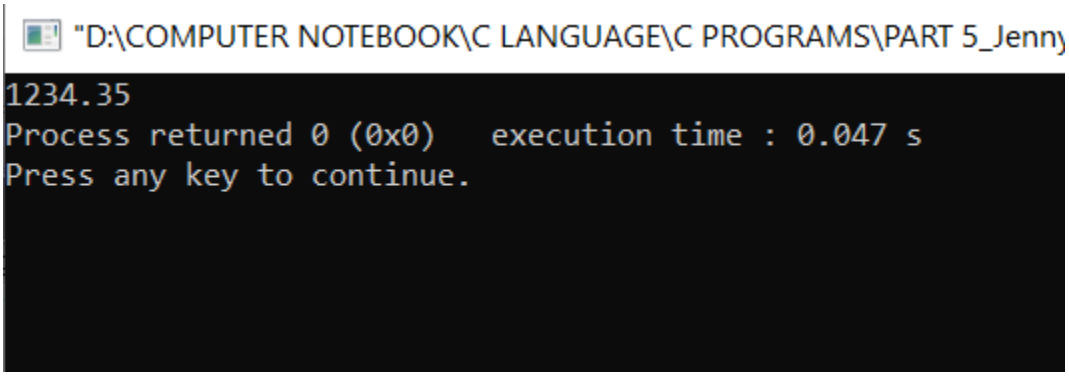
CODE 5:

```c
#include <stdio.h>
#include <stdlib.h>
/* 5 FORMATTED OUTPUT FUNCTIONS */
int main()
{
    float a=1234.3456;
    printf("%10.2f",a);
 /* 10 spaces of memory are alloted.
    In floating, point is also included in memory.
    with With point 9 values are there but
    floating precession is given as .2f
    So after decimal point 2 digits are precessed
    So now 1234.34, with decimal point 7 values are there
    So  7 values are printed within 10 memory spaces
    Justified from right, 3 spaces are blank*/
}
```



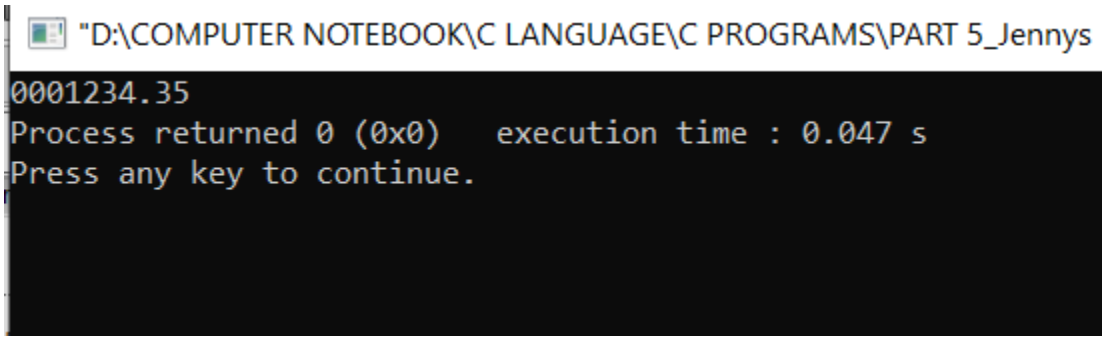
CODE 6:

```c
#include <stdio.h>
#include <stdlib.h>
/* 6 FORMATTED OUTPUT FUNCTIONS */
int main()
{
    float a=1234.3456;
    printf("%-10.2f",a);
 /* 10 spaces of memory are alloted.
    In floating, point is also included in memory.
    with With point 9 values are there but
    floating precession is given as .2f
    So after decimal point 2 digits are precessed
    So now 1234.34, with decimal point 7 values are there
    So  7 values are printed within 10 memory spaces
  %-10.2f Justified from left, last 3 spaces are blank*/
}
```



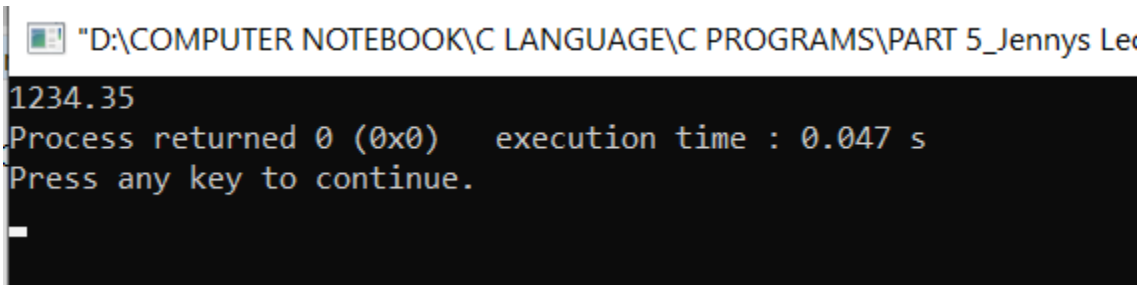
CODE 7:

```c
#include <stdio.h>
#include <stdlib.h>
/* 7 FORMATTED OUTPUT FUNCTIONS */
int main()
{
    float a=1234.3456;
    printf("%010.2f",a);
 /* 10 spaces of memory are alloted.
    In floating, point is also included in memory.
    with With point 9 values are there but
    floating precession is given as .2f
    So after decimal point 2 digits are precessed
    So now 1234.34, with decimal point 7 values are there
    So  7 values are printed within 10 memory spaces
  %10.2f Justified from right, first 3 spaces are Zeros*/
}
```



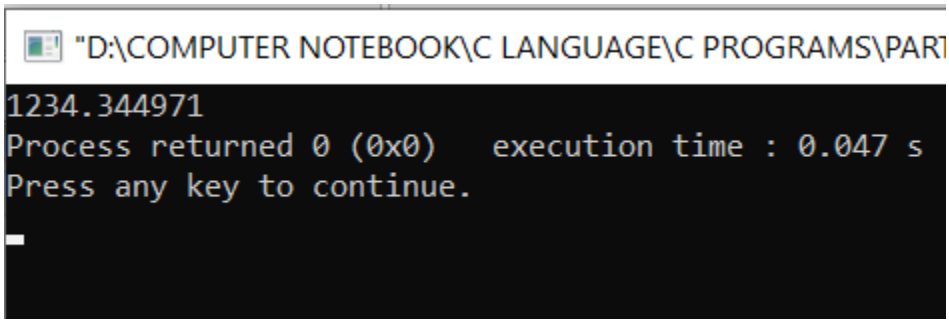
CODE 8:

```c
#include <stdio.h>
#include <stdlib.h>
/* 8 FORMATTED OUTPUT FUNCTIONS */
int main()
{
    float a=1234.3456;
    printf("%-010.2f",a);
 /* 10 spaces of memory are alloted.
    In floating, point is also included in memory.
    with With point 9 values are there but
    floating precession is given as .2f
    So after decimal point 2 digits are precessed
    So now 1234.34, with decimal point 7 values are there
    So  7 values are printed within 10 memory spaces
  %-10.2f Justified from LEFT,
  LAST 3 spaces are NOT Zeros because .2f is precessed*/
}
```



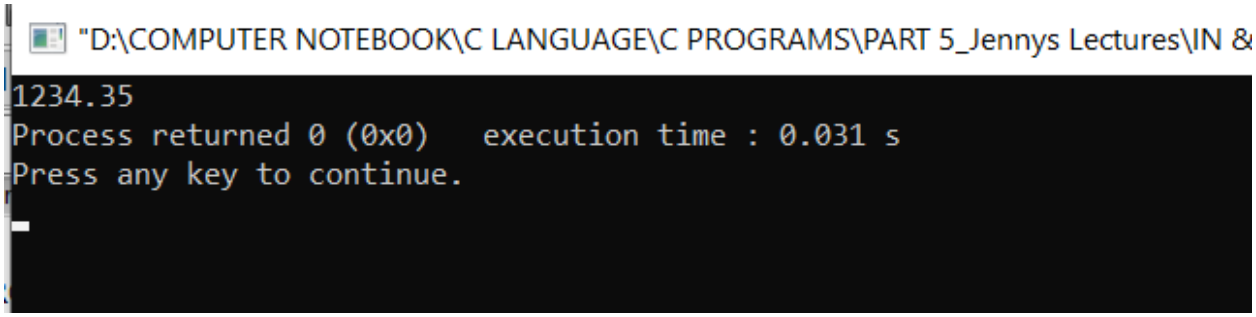
CODE 9:

```c
#include <stdio.h>
#include <stdlib.h>
/* 9 FORMATTED OUTPUT FUNCTIONS */
int main()
{
    float a=1234.345;
    printf("%f",a);
  /* Here float will precessed with 6 digits after decimal
  point */
}
```



CODE 10:

```c
#include <stdio.h>
#include <stdlib.h>
/* 10 FORMATTED OUTPUT FUNCTIONS */
int main()
{
    float a=1234.3456;
    printf("%5.2f",a);
  /* Even though we 5 memory spaces are alloted, it will
  print all the values but it will only  justify since
  it is formatted output not formatted input function */


}
```



CODE 11:

```c
#include <stdio.h>
#include <stdlib.h>
/* 11 FORMATTED OUTPUT FUNCTIONS */
int main()
{
    float a=1234.3456;
    printf("%e",a);
}

```