

COP 3223: C Programming Spring 2009

Introduction To C - Part 3

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
<http://www.cs.ucf.edu/courses/cop3223/spr2009/section1>

School of Electrical Engineering and Computer Science
University of Central Florida



More Basic C Programming

- Returning briefly to the program from the previous set of notes that produced the sum of two integer values, I want to show you a slight modification to the code that we could have made, which is perfectly legal in C.
- The modification involves the need and/or use of the variable `sum`. In the version of the program in the previous set of notes, we defined a variable, called `sum`, to hold the sum of `integer1 + integer2`.
- As is shown in the code on the next page, we didn't really need this variable, as we could have moved the calculation of the sum into the `printf` function call.



```
// sum of two integers (a second C program)
// This program adds two, user supplied, integers and prints their sum
// January 13, 2009   Written by: Mark Llewellyn

#include <stdio.h>

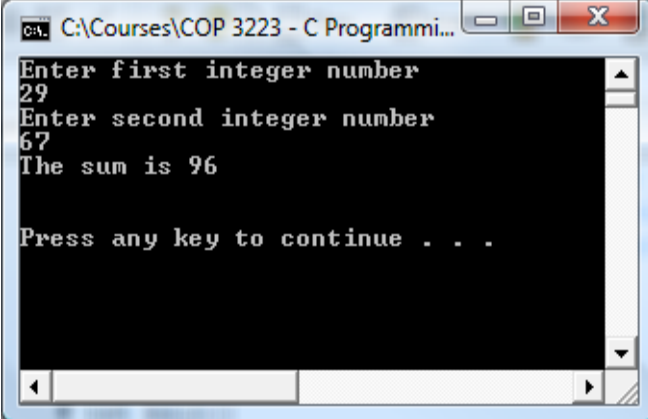
//main function
int main()
{
    int integer1; //first integer to be entered by user
    int integer2; //second integer to be entered by user

    //write prompts to user and get numbers
    printf("Enter first integer number\n");
    scanf("%d", &integer1);
    printf("Enter second integer number\n");
    scanf("%d", &integer2);

    printf("The sum is %d\n", integer1 + integer2);
    printf("\n\n");
    system("PAUSE");

    return 0;
} //end main function
```

Modified version of the
sum of two integers
program



A screenshot of a Windows command prompt window titled "C:\Courses\COP 3223 - C Programmi...". The window shows the execution of a C program. The output is as follows:

```
Enter first integer number
29
Enter second integer number
67
The sum is 96

Press any key to continue . . .
```



A Brief Aside On Data Types In C

- In the last section of notes, when we introduced the concept of a **variable**, we said that every variable is required to have a **name** and a **type**.
- There are many different data types in C, but for right now, we are restricting ourselves to two types: `int` (short for integer) and `float` (short for floating-point).
- We'll later see that correctly choosing the type of a variable is important from a program efficiency point of view, as well as determining what kinds of operations can be performed on the variable.
- The type of a numeric variable (a variable capable of storing a number) determines the smallest and largest value that can be stored in a variable of that type.



A Brief Aside On Data Types In C

Data type	Smallest value	Largest value
<code>short int</code>	-32,768	32,767
<code>unsigned short int</code>	0	65,535
<code>int</code>	-32,768	32,767
<code>unsigned int</code>	0	65,535
<code>long int</code>	-2,147,483,648	2,147,483,647
<code>unsigned long int</code>	0	4,294,967,295

The `int` data type on a 16-bit machine

A machine with a 16-bit word size means that a “word” (the addressable component of the machine’s memory) contains 16 bits. $2^{16} = 65,536$ values, not counting 0, so the maximum unsigned integer value would be 65,535. Signed integers require one of the bits for the sign, so only 15 bits are available for the number. $2^{15} = 32,768$, again counting for the zero value, this leaves only 32,767 possible positive and 32,768 negative values. Long integers, sometimes referred to as double precision require 2 words, or 32 bits. So, $2^{32} = 4,294,967,295$ possible unsigned long integer values. Similarly, signed long integers lose 1 bit of precision for the sign and thus, $2^{31} = 2,147,483,648$ possible negative and 2,147,483,647 positive values.



A Brief Aside On Data Types In C

Data type	Smallest value	Largest value
<code>short int</code>	-32,768	32,767
<code>unsigned short int</code>	0	65,535
<code>int</code>	-2,147,483,648	2,147,483,647
<code>unsigned int</code>	0	4,294,967,295
<code>long int</code>	-2,147,483,648	2,147,483,647
<code>unsigned long int</code>	0	4,294,967,295

The `int` data type on a 32-bit machine

$$2^{32} = 4,294,967,295$$



A Brief Aside On Data Types In C

Data type	Smallest value	Largest value
<code>short int</code>	-32,768	32,767
<code>unsigned short int</code>	0	65,535
<code>int</code>	-2,147,483,648	2,147,483,647
<code>unsigned int</code>	0	4,294,967,295
<code>long int</code>	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
<code>unsigned long int</code>	0	18,446,744,073,709,551,615

The `int` data type on a 64-bit machine

$$2^{64} = 18,446,744,073,709,551,616$$



Formatting Output With `printf`

- In the two previous sections of notes, we've developed programs that both accepted input from the user using the `scanf` function and printed output to the screen using the `printf` function.
- Let's take a closer look at the formatting capabilities possible with both of these functions, focusing first on the `printf` function.
- Precise formatting is accomplished with `printf`.
- Every `printf` call contains a **format control string** that describes the output format. (Recall that we looked at the format control string last time in the context of a `scanf` call.)
- The format control string consists of **conversion specifiers, flags, field widths, precisions, and literal characters**. Together with the percent sign (`%` - used as a special delimiter), these form the **conversion specifications**.



Formatting Output With `printf`

- The `printf` function can perform the following formatting capabilities:
 1. **Rounding** floating-point (real) values to an indicated number of decimal places.
 2. **Aligning** a column of numbers with decimal points appearing one above the other.
 3. **Right-justification** and **left justification** of outputs.
 4. Inserting **literal characters** at precise locations in a line of output.
 5. Representing **floating-point numbers** in exponential format.
 6. Displaying all types of data with fixed-size field widths and precisions.



Formatting Output With `printf`

- The `printf` function has the following general form:

```
printf( format-control-string, other-arguments);
```

- Even though we already seen some of the conversion specifiers in the previous set of notes, let's look at them again, this time in the context of a `printf` call and distinguish amongst the various types of data that we can print.



Conversion Specifiers for Integers

Conversion Specifier	Description
d	Display a signed decimal integer.
i	Display a signed decimal integer. (Note the i and d specifiers are different when used with <code>scanf</code> .)
o	Display an unsigned octal integer (Base 8 number).
u	Display an unsigned decimal integer.
x or X	Display an unsigned hexadecimal integer. X causes the digits 0-9 and letters A-F to be displayed and x causes the digits 0-9 and the letters a-f to be displayed.
h or l	Place before any integer conversion specifier to indicate that a short or long integer is displayed respectively. The letters h and l are more precisely called length modifiers .



```
printf integer examples.c - Notepad
File Edit Format View Help
// printf - integer examples
// January 14, 2009

#include <stdio.h>

int main()
{
    printf( "%d\n", 455 );
    printf( "%i\n", 455 ); // i same as d in printf
    printf( "%d\n", +455 );
    printf( "%d\n", -455 );
    printf( "%hd\n", 32000 );
    printf( "%ld\n", 2000000000 );
    printf( "%o\n", 455 );
    printf( "%u\n", 455 );
    printf( "%u\n", -455 );
    printf( "%x\n", 455 );
    printf( "%X\n", 455 );

    printf("\n\n");
    system("PAUSE");

    return 0;
} // end main function
```

It is an "error" to print a signed integer using an unsigned specifier. The value seen is essentially garbage.

```
C:\Courses\COP 3223 - C Programming\S...
455
455
455
-455
32000
2000000000
707
455
4294966841
1c7
1C7

Press any key to continue . . .
```



Printing Floating-Point Numbers

- A floating-point value contains a decimal point as in 77.4, 69.69, or -743.293.
- Floating-point values are commonly displayed in one of several formats, typically, depending either on the magnitude of the number or the desired form for an application.
- The table on the next page illustrates the conversion specifiers that are applicable to floating-point values.



Conversion Specifiers for Floating-Point Values

Conversion Specifier	Description
<code>e</code> or <code>E</code>	Display floating-point values in exponential notation. <code>e</code> prints a lowercase <code>e</code> for the exponential part while <code>E</code> prints an uppercase <code>E</code> for the exponential part.
<code>f</code>	Display floating-point values in fixed-point notation.
<code>g</code> or <code>G</code>	Display a floating-point value in either the floating-point form <code>f</code> or the exponential form <code>e</code> (or <code>E</code>) based on the magnitude of the value.
<code>L</code>	Place before any floating-point conversion specifier to indicate that a long double floating-point value is displayed.

Values printed by the `e`, `E`, or `f` specifiers are output with six digits of precision to the right of the decimal point by default (e.g., 3.141596).

Conversion specifier `f`, always prints at least one digit to the left of the decimal point.

Specifiers `e` and `E` always print 1 digit to the left of the decimal point.



```
printf float examples.c - Notepad
File Edit Format View Help
// printf - float - examples
// January 14, 2009

#include <stdio.h>

int main()
{
    printf( "%e\n", 1234567.89 );
    printf( "%e\n", +1234567.89 );
    printf( "%e\n", -1234567.89 );
    printf( "%E\n", 1234567.89 );
    printf( "%f\n", 1234567.89 );
    printf( "%g\n", 1234567.89 );
    printf( "%G\n", 1234567.89 );
    printf("\n\n");
    system("PAUSE");

    return 0;
} // end main function
```

```
C:\Courses\COP 3223 - C Programmin...
1.234568e+006
1.234568e+006
-1.234568e+006
1.234568E+006
1234567.890000
1.23457e+006
1.23457E+006

Press any key to continue . . .
```



Printing With Field Widths

- The exact size of a field in which data is to be printed is specified by a **field width**.
- If the field width is larger than the data being printed, the data is normally right-justified (i.e., pushed as far to the right in the field as possible) within the field.
- An integer representing the field width is inserted between the percent sign (%) and the conversion specifier (e.g., %6d).
- The sample program on the next page illustrates the use of field widths and integer values.



Using Field Widths

```
printf - field width examples.c - Notepad
File Edit Format View Help
// printf - field width - examples
// January 14, 2009

#include <stdio.h>

int main()
{
    printf( "%4d\n", 1 );
    printf( "%4d\n", 12 );
    printf( "%4d\n", 123 );
    printf( "%4d\n", 1234 );
    printf( "%4d\n\n", 12345 ); // data too large for field

    printf( "%4d\n", -1 );
    printf( "%4d\n", -12 );
    printf( "%4d\n", -123 );
    printf( "%4d\n", -1234 ); // data too large for field
    printf( "%4d\n", -12345 ); // data too large for field
    printf( "\n\n");
    system("PAUSE");

    return 0;
} // end main function
```

```
C:\Courses\COP 3223 - C Programmin...
1
 12
 123
1234
12345

-1
-12
-123
-1234
-12345

Press any key to continue . . .
```



Printing With Precision

- The `printf` function also provides the ability to specify the **precision** with which data is printed.
- Precision has different meanings for different data types.
- When used with integer conversion specifiers, precision indicates the minimum number of digits to be printed. If the printed value contains fewer digits than the specified precision, zeros are prefixed to the printed value until the total number of digits is equal to the precision value specified. The default precision is 1.
- When used with floating-point conversion specifiers `e`, `E`, and `f`, the precision is the number of digits to appear to the right of the decimal point.



Printing With Precision

- When used with conversion specifiers `g` and `G`, the precision is the maximum number of significant digits to be printed.
- When used with the conversion specifier `s` (for strings), the precision is the maximum number of characters to be printed from the string.
- To use precision, place a decimal point (.) followed by an integer representing the precision between the percent sign (%) and the conversion specifier, e.g., `%.4d`.
- The sample program on the next page illustrates the use of precision with conversion specifiers.



Using Precision

printf - precision examples.c - Notepad

File Edit Format View Help

```
// printf using precision
// January 15, 2009

#include <stdio.h>

int main()
{
    int i = 873;           //initialize int i
    double f = 123.94536; // initialize double f

    printf("Using precision for integers\n" );
    printf("\t%.4d\n", i);
    printf("\t%.9d\n\n", i);

    printf("Using precision for floating-point numbers\n" );
    printf("\t%.3f\n", f);
    printf("\t%.3e\n", f);
    printf("\t%.3g\n", f);
    printf("\n Note that you can also print all three from one printf statement\n");
    printf("\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
    system("PAUSE");

    return 0;
}
```



Using precision for integers

```
0873
000000873
```

Using precision for floating-point numbers

```
123.945
1.239e+002
124
```

Note that you can also print all three from one printf statement

```
123.945
1.239e+002
124
```

Press any key to continue . . .

Note: when a floating-point value is printed with a precision smaller than the original number of decimal places in the value, the value is rounded.



Combining Field Widths And Precision

- A very common occurrence is to use both field width and precision when using the `printf` function.
- To use both field width and precision, place the field width first, followed by a decimal point (.) followed by an integer representing the precision between the percent sign (%) and the conversion specifier, e.g., `%9.3d`.
- If we added the statement `printf("%9.3f", 123.456789)` to the previous program, it would display 123.457, with three digits to the right of the decimal point right-justified in a nine-digit field.



Combining Field Widths And Precision

There is also another special way that field widths and precision can be used together by using an integer expression in the argument list following the format control string. To use this feature, insert an asterisk (*) in place of the field width or precision (or both) in the format control string. The matching `int` argument in the argument list is evaluated and used in place of the asterisk. A field width's value may be either positive or negative (which causes the output to be left-justified in the field)

The statement `printf("%*.*f", 7, 2, 69.866);`
Uses 7 for the overall field width and 2 for the precision and would thus print the value 69.87 right-justified.



Using Flags In The `printf` Format Control String

- The function `printf` also provides flags to supplement its output formatting capabilities.
- There are five flags available as shown in the table on the next page.
- To use a flag in a format control string, place the flag immediately to the right of the percent sign. Several flags can be combined in a one conversion specifier.
- A series of sample programs beginning on page 26, illustrate the use of flags in the format control string of a `printf` function call.



Format Control String Flags

Flag	Description
- (minus sign)	Left-justify the output within the specified field width.
+ (plus sign)	Display a plus sign preceding positive values and a minus sign preceding negative values
space	Print a space before a positive value not printed with the + flag.
#	Prefix 0 to the output value when used with the octal conversion specifier o . Prefix 0x or 0X to the output values when used with the hexadecimal conversion specifiers x or X . Force a decimal point for a floating-point number printed with e, E, f, g, or G that does not contain a fractional part (default only prints decimal point if a number follows it). For g and G specifiers, trailing zeros are not eliminated.
0 (zero)	Pad a field with leading zeros.



Using Flags

Left and Right Justification

```
printf - flags.c - Notepad
File Edit Format View Help
// printf using flags - left-justify
// January 15, 2009
#include <stdio.h>
int main()
{
    printf("\n%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23);
    printf("%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23);
    printf("\n\n");
    system("PAUSE");
    return 0;
} // end main function
```

Minus sign flag causes left-justification

```
C:\Courses\COP 3223 - C Programming\Sprin...
hello      7      a 1.230000
hello      7      a      1.230000
Press any key to continue . . .
```



Using Flags

+ Flag

```
printf - plus flag.c - Notepad
File Edit Format View Help
//printf using flags - with and without + flag
//January 15, 2009

#include <stdio.h>

int main()
{
    printf("\n");
    printf("%d\n%d\n", 786, -786 );
    printf("%+d\n%+d\n", 786, -786 );
    printf("\n\n");
    system("PAUSE");

    return 0;

} // end main function
```

```
C:\Courses\COP 3223 - C Progra...
786
-786
+786
-786

Press any key to continue . . .
```



Using Flags

<space> Flag

```
printf - space flag.c - Notepad
File Edit Format View Help
//printf - space flag usage
//January 15, 2009

#include <stdio.h>

int main()
{
    printf("\n");
    printf(" %d\n% d\n", 547, -547 );
    printf("\n\n");
    system("PAUSE");

    return 0;
} // end main function
```

```
C:\Courses\COP 3223 - C Programmin...
547
-547

Press any key to continue . . .
```



Using Flags

Flag

```
printf - pound flag .c - Notepad
File Edit Format View Help
//printf - # flag example
//January 15, 2009

#include <stdio.h>

int main()
{
    int c = 1427;    //initialize c
    double p = 1427.0; // initialize p

    printf("\n");
    printf("%#o\n", c );
    printf("%#x\n", c );
    printf("%#X\n", c );
    printf("\n%g\n", p );
    printf("%#g\n", p );
    printf("\n\n");
    system("PAUSE");

    return 0;
} // end main function
```

```
C:\Courses\COP 3223 - C Program...
02623
0x593
0X593

1427
1427.00

Press any key to continue . . . .
```



Using Flags

0 (zero) Flag

```
printf - zero flag.c - Notepad
File Edit Format View Help
//printf - zero flag example
//January 15, 2009

#include <stdio.h>

int main()
{
    printf("\n");
    printf("%+09d\n", 452 );
    printf("%09d\n", 452 );
    printf("\n\n");
    system("PAUSE");

    return 0;
} // end main function
```

```
C:\Courses\COP 3223 - C Progra...
+000000452
000000452

Press any key to continue . . .
```



Formatting Input Using `scanf`

- Precise input formatting can be accomplished using the `scanf` function. Although it is not quite as common to quite formatted input as it is to use formatted output, in certain cases it is a nice feature to be able to use.
- Every `scanf` statement contains a format control string that describes the format of the data to be input (see page 3 of this set of notes for an example).
- For the `scanf` function, the format control string consists of conversion specifiers and literal characters.



Formatting Input Using `scanf`

- The `scanf` function can provide the following input formatting capabilities:
 1. Inputting all types of data.
 2. Inputting specific characters from an input screen.
 3. Skipping specific characters in the input screen.
- The general form of a `scanf` function call is:

```
scanf( format-control-string, other-arguments );
```

The *format-control-string* describes the formats of the input, and the *other arguments* are points to variables in which the input will be stored.



Conversion Specifiers `scanf`

Conversion Specifier	Description
<code>d</code>	Read an optionally signed decimal integer. The corresponding argument is a pointer to an integer variable.
<code>i</code>	Read an optionally signed decimal, octal or hexadecimal integer. The corresponding argument is a pointer to an integer.
<code>o</code>	Read an octal integer. The corresponding argument is a pointer to an unsigned integer.
<code>u</code>	Read an unsigned decimal integer. The corresponding argument is pointer to an unsigned integer variable.
<code>x</code> or <code>X</code>	Read a hexadecimal integer. The corresponding argument is a pointer to an unsigned integer.
<code>h</code> or <code>l</code>	Place before any of the integer conversion specifiers to indicate that a <code>short</code> or <code>long</code> integer is to be input.

`scanf` conversion specifiers for use with integer values



Conversion Specifier	Description
<code>e, E, f, g, or G</code>	Read a floating-point value. The corresponding argument is a pointer to an floating-point variable.
<code>l or L</code>	Place before any of the floating-point conversion specifiers to indicate that a <code>double</code> or <code>long double</code> value is to be input. The corresponding argument is a point to a <code>double</code> or <code>long double</code> variable.

`scanf` conversion specifiers for use with floating-point values

Conversion Specifier	Description
<code>c</code>	Read a character. The corresponding argument is a pointer to a <code>char</code> ; no null (<code>'\0'</code>) is added.
<code>s</code>	Read a string. The corresponding argument is a pointer to an array of type <code>char</code> that is large enough to hold the string and a terminating null (<code>'\0'</code>) character – which is automatically added.

`scanf` conversion specifiers for use with characters and strings



```
scanf - integer specifiers.c - Notepad
File Edit Format View Help
// scanf with integer conversion specifiers
//January 15, 2009

#include <stdio.h>

int main()
{
    int a;
    int b;
    int c;
    int d;
    int e;
    int f;
    int g;

    printf("\n");
    printf("Enter seven integers: " );
    scanf("%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );

    printf( "The input displayed as decimal integers is:\n" );
    printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );
    printf("\n\n");
    system("PAUSE");

    return 0;
} // end main function
```

scanf with integer conversion specifiers

```
C:\Courses\COP 3223 - C Programming\Spring 20...
Enter seven integers: 2 3 4 5 6 7 8
The input displayed as decimal integers is:
2 3 4 5 6 7 8

Press any key to continue . . . _
```

```
C:\Courses\COP 3223 - C Programming\Spring 2009\C...
Enter seven integers: -70 -70 070 0x70 70 70 70
The input displayed as decimal integers is:
-70 -70 56 112 56 70 112

Press any key to continue . . . _
```



scanf with floating-point conversion specifiers

scanf - floatingpoint specifiers.c - Notepad

File Edit Format View Help

```
//scanf using floating-point conversion specifiers
//January 15, 2009

#include <stdio.h>

int main()
{
    double a;
    double b;
    double c;

    printf("\n");
    printf("Enter three floating-point numbers: \n" );
    scanf("%le%lf%lg", &a, &b, &c );
    printf("\nHere are the numbers entered in plain\n" );
    printf("floating-point notation:\n" );
    printf("%f\n%f\n%f\n", a, b, c );
    printf("\n\n");
    system("PAUSE");

    return 0;
} // end main function
```

C:\Courses\COP 3223 - C Programming\Spring ...

```
Enter three floating-point numbers:
23.45 123.699 456.98823

Here are the numbers entered in plain
floating-point notation:
23.450000
123.699000
456.988230

Press any key to continue . . . .
```

C:\Courses\COP 3223 - C Programming\S...

```
Enter three floating-point numbers:
1.69699 1.69699e+04 3.14159e-06

Here are the numbers entered in plain
floating-point notation:
1.696990
16969.900000
0.000003

Press any key to continue . . . .
```



scanf using field widths

```
scanf - field width.c - Notepad
File Edit Format View Help
//scanf using a field width
//January 15, 2009

#include <stdio.h>

int main()
{
    int x;
    int y;

    printf("\n");
    printf("Enter a six digit integer: ");
    scanf("%2d%d", &x, &y );
    printf("\nThe integers input were %d and %d\n", x, y );
    printf("\n\n");
    system("PAUSE");

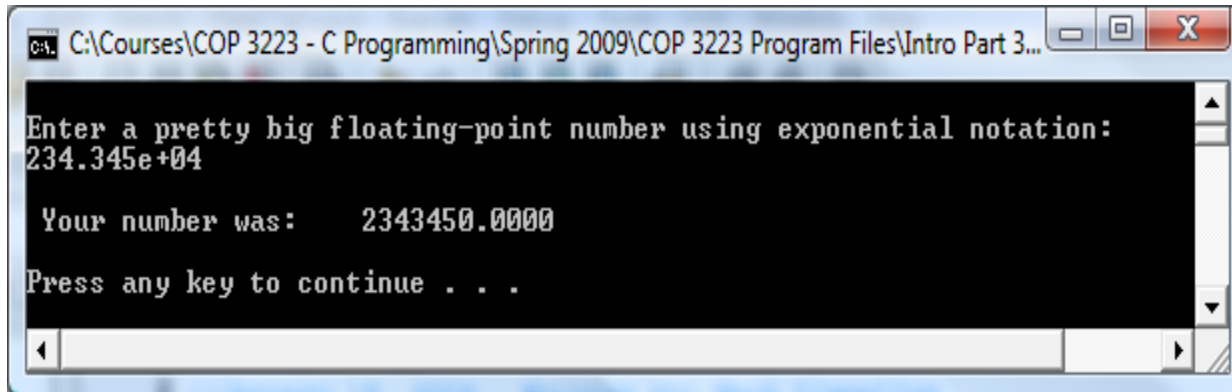
    return 0;
} // end main function
```

```
C:\Courses\COP 3223 - C Programmin...
Enter a six digit integer: 123456
The integers input were 12 and 3456
Press any key to continue . . .
```



Practice Problems

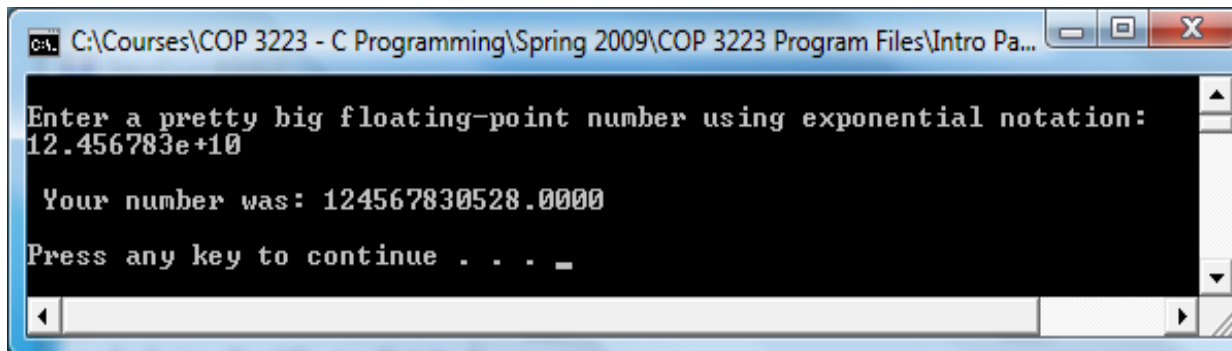
1. Write a C program that will ask the user to enter a very large floating-point number using exponential notation and the program will then print that number in normal decimal notation. Assume that the very large number used for the input would not require a field larger than 15 digits.



```
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files\Intro Part 3...
Enter a pretty big floating-point number using exponential notation:
234.345e+04

Your number was:    2343450.0000

Press any key to continue . . .
```



```
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files\Intro Pa...
Enter a pretty big floating-point number using exponential notation:
12.456783e+10

Your number was: 124567830528.0000

Press any key to continue . . . _
```

