

C: From Theory to Practice

Chapter 2 *Data Types, Variables and Data Output*

George S. Tselikis and Nikolaos D. Tselikas

Variables

♦ RAM and Variables

- The computer's RAM (Random Access Memory) consists of millions of successive storage cells
- The size of each cell is one **byte**
- For example, an old PC with 16 MB (megabytes) of RAM consists of:
 $16 \times 1024 \text{ kB (kilobytes)} =$
 $16.777.216 \text{ memory cells}$
- A newer PC with 8 GB (gigabytes) of RAM would have:
 $8 \times 1024 \text{ MB} = 8192 \times 1024 \text{ kB} =$
 $8.388.608 \times 1.024 = 8.589.934.592 \text{ memory cells}$
- A **variable** in C is a **storage location** with a **given name**
- The **value** of a variable is the **content** of its **memory location**, while a program may **use the name** of a variable **to access** its value

Rules for Naming Variables

- ◆ Be sure to apply the basic rules for naming variables or your code won't compile
 - The name of a variable can contain **uppercase letters, lowercase letters, digits** and the **underscore** character `'_'`
 - The name must begin with either a **letter** or the **underscore** character `'_'`
 - The C programming language is **case sensitive**, meaning that it distinguishes between uppercase and lowercase letters
 - ◆ For example, the variable `temp` is different from `Temp` or `tEmP`
 - The **keywords** of C language **can not be used** as variable names because they have special significance to the C compiler

C keywords

auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	
continue	for	return	typedef	
default	float	short	union	

Remarks

- Use descriptive names for variables, since it's much easier to read a program when the names of the variables indicate their intended use
 - ◆ E.g. if you plan to use a variable to hold the sum of some even numbers, name that variable something like `sum_even` rather than an arbitrary name like `i`
- When necessary, don't be afraid to use long names to describe the role of a variable and if a variable name is several words long, separate each word with the underscore character (`_`) for readability
 - ◆ E.g. you might call a variable that holds the number of books in a calculation `books_number` (instead of `booksnumber`, or something less readable)
- By convention (this is not a requirement) C programmers use lower-case letters when naming variables and upper-case letters when defining macros and constants

Declaring Variables

- Variables must be declared before being used in a program
- Declare a variable as follows:

```
data_type name_of_variable;
```

- The `name_of_variable` is the variable name, while the `data_type` should be one of the C supported data types
 - ♦ E.g., the `int` keyword is used to declare integer variables, and the `float` keyword is used to declare floating point variables, i.e. variables that can store values with a fractional part

C Data Types

Type	Size (bytes)	Range (min – max)
<code>char</code>	1	-128 ... 127
<code>short</code>	2	-32.768 ... 32.767
<code>int</code>	4	-2.147.483.648...2.147.483.647
<code>long</code>	4	-2.147.483.648...2.147.483.647
<code>float</code>	4	Lowest positive value: $1.17 \cdot 10^{-38}$ Highest positive value: $3.4 \cdot 10^{38}$
<code>double</code>	8	Lowest positive value: $2.2 \cdot 10^{-308}$ Highest positive value: $1.8 \cdot 10^{308}$
<code>long double</code>	8, 10, 12, 16	
<code>unsigned char</code>	1	0 ... 255
<code>unsigned short</code>	2	0 ... 65535
<code>unsigned int</code>	4	0 ... 4.294.967.295
<code>unsigned long</code>	4	0 ... 4.294.967.295

Examples and Remarks

```
int a; /* Declare an integer variable with name a. */  
float b; /* Declare a float variable with name b. */
```

- Variables of the same type can be declared in the same line, separated with a comma (,)
 - ♦ So, instead of declaring the variables `a`, `b` and `c` in three different lines:

```
int a;  
int b;  
int c;
```

you can declare them in a single line, as:

```
int a, b, c;
```

- The memory space that a data type requires may vary from one system to another
 - ♦ For example, the `int` type may reserve 2 bytes in one system and 4 bytes in another
 - ♦ To determine the number of bytes a data type uses on a particular system, use the `sizeof` operator (discussed in Chapter 4)
- If the precision of your data is not critical use the `float` type, because `float` usually reserves fewer bytes than `double`, and calculations with `float` numbers tend to be executed faster than with `double` numbers
- If the precision is critical, use the `double` type

Assigning Values to Variables (1/2)

- A variable can be given a value by using the assignment operator (=)

- ♦ E.g. the following statement assigns the value 100 to the `int` variable `a`

```
int a;  
a = 100;
```

- Alternatively, a variable can be initialized together with its declaration.

- ♦ E.g. the above statements could be replaced in one line, as:

```
int a = 100;
```

Assigning Values to Variables (2/2)

- You can also initialize more than one variable of the same type together with its declaration, e.g.

```
int a = 100, b = 200, c = 300;
```

- To assign a floating point value to a `float` variable we use the dot (.) for the fractional part and not the comma (,) e.g.

```
float a = 1.24;
```

- If an integer value begins with the digit 0, this value will be interpreted as an octal (base-8) number
 - ♦ E.g. the following statement assigns the decimal value 64 and not the value 100 to the variable `a`

```
int a = 0100;
```

- Similarly, a value that begins with `0x` or `0X` is interpreted as hexadecimal (base-16) number
 - ♦ E.g., the following statement assigns the decimal value 16 to variable `a`

```
int a = 0x10;
```

Remarks (1/2)

- The value assigned to a variable should be within the range of its type
 - ♦ E.g., the statement:

```
char ch = 130;
```

 - ♦ does not make the value of `ch` equal to 130, since the range of the values of the signed `char` type is from -128 to 127
 - ♦ 130 is out of the allowed range, so the value is wrapped around to -126
- A floating point value can be written in scientific notation using the letter `E` (or `e`), which represents the power of 10
 - ♦ E.g., instead of `a = 0.085;`
we can write `a = 85E-3;`
which is equivalent to $85 \cdot 10^{-3}$
 - ♦ E.g., `a = 1.56e6;`
is equivalent to `a = 1560000;` or $1.56 \cdot 10^6$

Remarks (2/2)

- The value assigned to a variable should match the variable type

- ♦ E.g., the statement:

```
int a = 10.9;
```

actually sets the value of `a` to 10, since `a` has been declared as `int` and not `float`



The decimal part is completely ignored and the assigned value is not rounded to 11

- However, the value of a `float` variable can be an integer

- ♦ E.g., you could write:

```
float a = 50;
```

since that's equivalent to:

```
float a = 50.0;
```

Constants

- A variable whose value can not change during the execution of the program is called a constant
- To declare a constant, precede the type of the variable with the `const` keyword
- A constant must be initialized when it is declared and you can not assign it another value within the program
 - ♦ E.g., the following statement declares the integer variable `a` as constant and sets it equal to 10

```
const int a = 10;
```
 - ♦ If we attempt to change the value of a constant later in a program, e.g., by writing:

```
a = 100;
```

the compiler will raise an error message

The #define Directive

- An alternative to declare a constant in a program is to use the `#define` directive, which is used to define a macro
- A macro is a name, which - in most cases - represents a numerical value
- To define a simple macro we write:

```
#define name_of_macro value
```

- When a program is compiled each macro is replaced by its defined value
- E.g.:

```
#define NUM 100
```

a macro called `NUM` with value `100` is defined

- When the program is compiled `NUM` is replaced by `100`

Remarks

- Macros are typically defined before the `main()` function, and are usually named using all capital letters



Note that there is no semicolon at the end of a `#define` directive

- In general, macros are most helpful when they're used to represent a numeric value that appears many times within your program

```
#include <stdio.h>

#define NUM 100

int main()
{
    int a,b,c;
    a = 20 - NUM;
    b = 20 + NUM;
    c = 3*NUM;
    return 0;
}
```

vs.

```
#include <stdio.h>

int main()
{
    int a, b, c;
    a = 20 - 100;
    b = 20 + 100;
    c = 3 * 100;
    return 0;
}
```

It's safer and faster (especially in programs with thousand of lines...)

The `printf()` function

- The `printf()` function is used to print a variable number of data items to the standard output stream (`stdout`), which, by default, is associated with the screen
- `printf()` accepts several parameters
 - ◆ The first (mandatory) parameter is a **format string**, that is a sequence of characters in double quotes (" "), which determines the output format
 - ◆ The next parameters are optional and, if any, `printf()` displays their values to the screen
- The format string may contain:
 - ◆ **Escape Sequences**
 - ◆ **Conversion Specifications**
 - ◆ **Ordinary characters** (which are printed as is to the screen)

Escape Sequences

- Escape sequences tell the compiler to perform a specific action, such as move the cursor
- An escape sequence consists of a backslash (\) followed by a character

Escape sequence	Action
\a	Make an audible beep.
\b	Delete the last character (equivalent to using the Backspace key).
\n	Advance the cursor to the beginning of the next line (equivalent to using the Enter key).
\r	Move the cursor to the beginning of the current line (equivalent to a carriage return).
\t	Move the cursor to the next tab stop (equivalent to the Tab key).
\\	Display a single backslash (\).
\"	Display double quotes (").

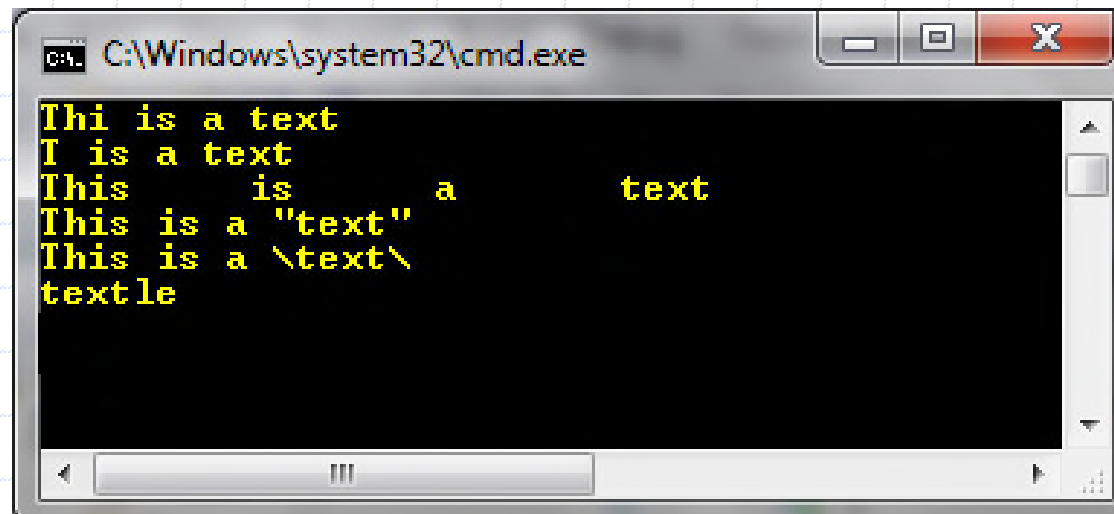
Conversion Specification

- Conversion specifications always begin with the percent (%) character and it is followed by one or more characters with special significance
- In its simplest form, a conversion specification is followed by one special character, called **conversion specifier**, listed in the table

Conversion Specifier	Meaning
c	Display the character which corresponds to an integer value.
d, i	Display a signed integer in decimal form.
u	Display an unsigned integer in decimal form.
f	Display a floating-point number in decimal form using a decimal point. The default precision is six digits after the decimal point.
s	Display a sequence of characters.
e, E	Display a floating-point number in scientific notation using an exponent. The exponent is preceded by the specifier.
g, G	Display a floating-point number either in decimal form (%f) or scientific notation (%e).
p	Display the value of a pointer variable.
x, X	Display an unsigned integer in hex form: %x displays lowercase letters (a-f), while %X displays uppercase letters (A-F).
o	Display an unsigned integer in octal.
%	Display the character %.

Examples (I)

```
#include <stdio.h>
int main()
{
    printf("\a");
    printf("This\b is a text\n");
    printf("This\b\b\b is a text\n");
    printf("This\t is\t a\t text\n");
    printf("This is a \"text\"\n");
    printf("This is a \\text\\n");
    printf("Sample\rtext\n");
    return 0;
}
```

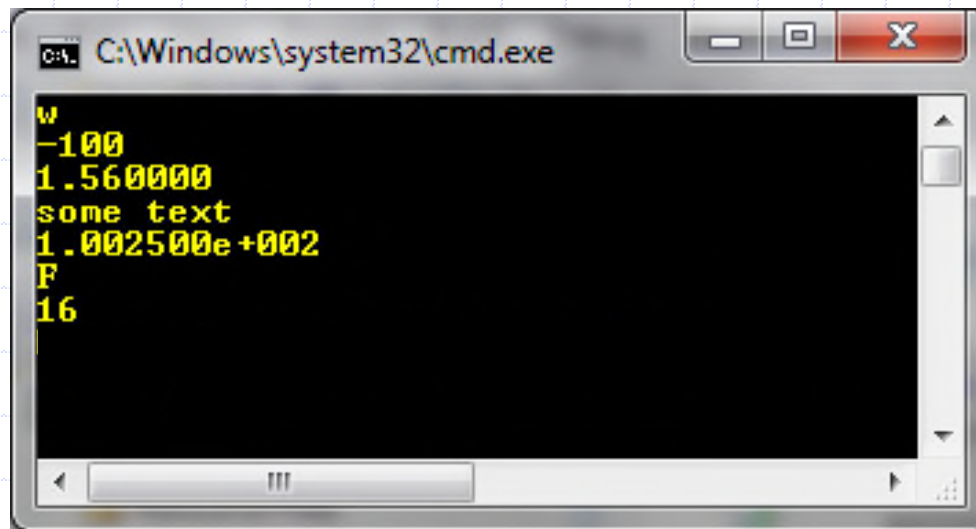


The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The output of the program is displayed in yellow text on a black background:

```
Thi is a text
I is a text
This   is   a       text
This is a "text"
This is a \text\
textle
```

Examples (II)

```
#include <stdio.h>
int main()
{
    printf("%c\n", 'w');
    printf("%d\n", -100);
    printf("%f\n", 1.56);
    printf("%s\n", "some text");
    printf("%e\n", 100.25);
    printf("%X\n", 15);
    printf("%o\n", 14);
    return 0;
}
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with yellow text. The output of the program is displayed line by line: 'w', -100, 1.560000, some text, 1.002500e+002, F, and 16. The window includes standard Windows window controls (minimize, maximize, close) in the top right corner and a scroll bar on the right side.

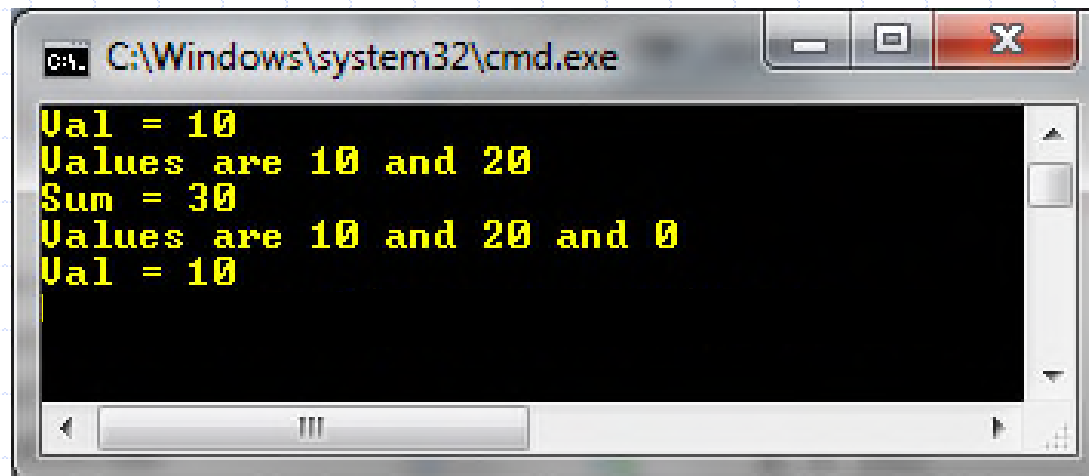
```
C:\Windows\system32\cmd.exe
w
-100
1.560000
some text
1.002500e+002
F
16
```

Printing Variables

- In `printf()` variable names follow the last double quote (" ") of the format string
- When printing more than one variable, separate each with a comma (,)
- The compiler will associate the conversion specifications with the names of each of the variables from left to right
- Each conversion specification should match the type of the respective variable, or the output will be meaningless

Example

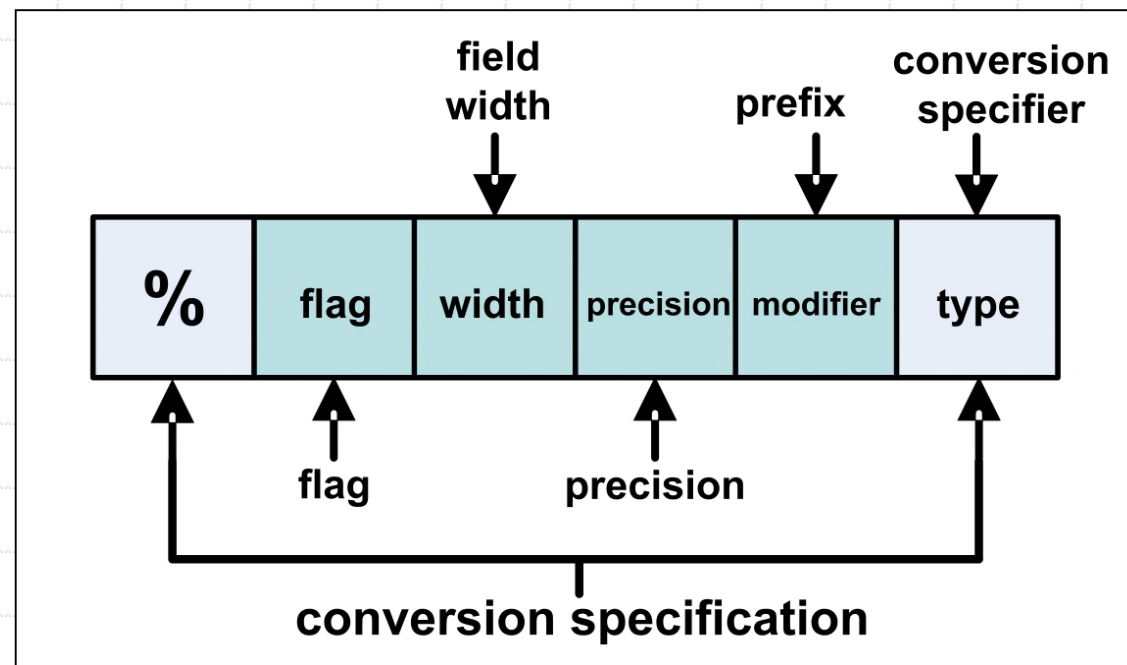
```
#include <stdio.h>
int main()
{
    int a,b;
    a = 10;
    b = 20;
    printf("Val = %d\n",a);
    printf("Values are %d and %d\n",a,b);
    printf("Sum = %d\n",a+b);
    printf("Values are %d and %d and %d\n",a,b);
    printf("Val = %d\n",a,b);
    return 0;
}
```

A screenshot of a Windows command prompt window. The title bar shows the path "C:\Windows\system32\cmd.exe". The window contains the following output in yellow text on a black background:

```
Val = 10
Values are 10 and 20
Sum = 30
Values are 10 and 20 and 0
Val = 10
```

Optional Fields

- The simplest form of the conversion specification begins with the % character followed by the conversion specifier
- However, a conversion specification may include another four fields as shown in the figure



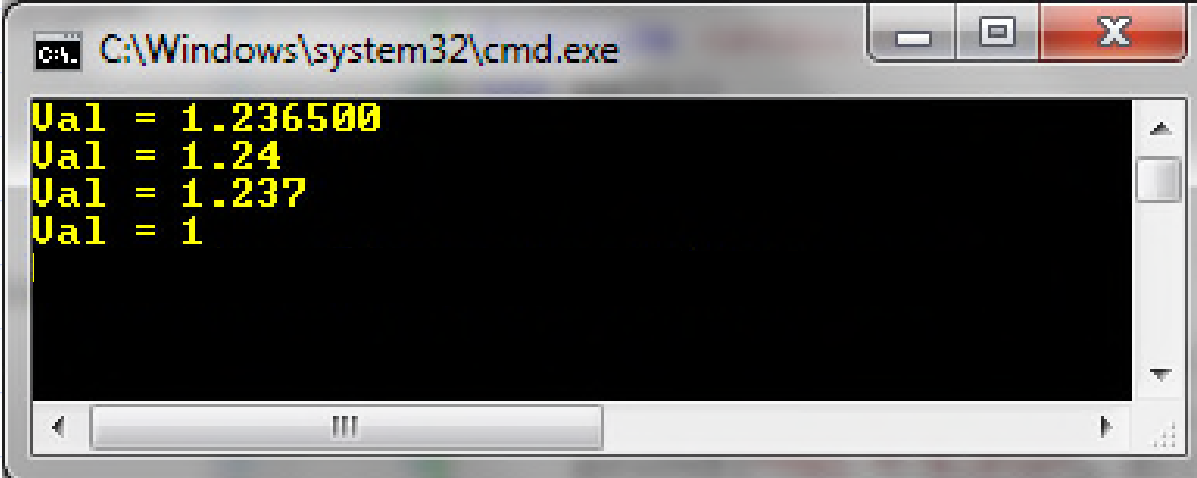
Precision

- When displaying the value of a floating-point type (i.e., `float` or `double`), we can specify the number of significant digits
- The default precision is six digits
- To specify another precision add a period (.) followed by
 - ◆ either an integer (to specify a precise number)
 - ◆ or an asterisk (*) and in this case the precise number of significant digits is defined by the next argument
- If the precision digits are less than the decimal digits, the displayed value is rounded up or down, according to the value of the cut-off digit
 - ◆ If it is less than 5 the displayed value is rounded down
 - ◆ otherwise it is rounded up
- To display no significant digits, add only a period (.)

Example

```
#include <stdio.h>
int main()
{
    float a = 1.2365;

    printf("Val = %f\n", a);
    printf("Val = %.2f\n", a);
    printf("Val = %.3f\n", a);
    printf("Val = %.f\n", a);
    return 0;
}
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The command prompt shows the output of the C program, with each line of output on a new line:

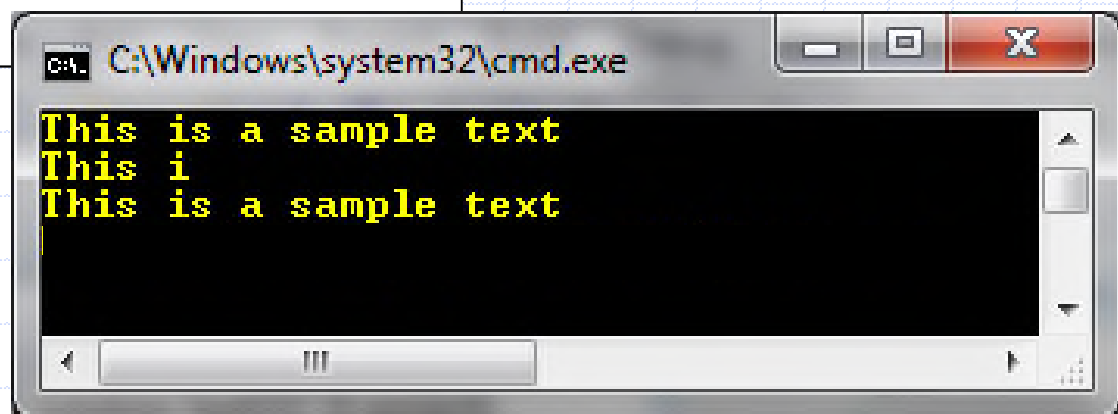
```
Val = 1.236500
Val = 1.24
Val = 1.237
Val = 1
```

The output demonstrates the effect of different format specifiers in the printf function: %f (full precision), %.2f (two decimal places), %.3f (three decimal places), and %.f (integer part only).

Display characters in strings

- When displaying a string we can define how many of its characters will be displayed as with floating-point numbers
- If the defined precision exceeds the number of the string's characters, the string is displayed as is

```
#include <stdio.h>
int main()
{
    char msg[] = "This is a sample text";
    printf("%s\n",msg);
    printf("%.6s\n",msg);
    printf("%.30s\n",msg);
    return 0;
}
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window contains the following output from the C program:

```
This is a sample text
This i
This is a sample text
```

The output demonstrates the effect of the printf format specifiers: the first line is the full string, the second line is truncated to 6 characters, and the third line is the full string again because the precision (30) exceeds the string's length.

Field Width

- When displaying the value of an integer or floating-point variable we can define the total number of characters to be displayed by adding
 - ◆ an integer (to specify the width of the output field)
 - ◆ or an asterisk (*) (in which case the width is defined by the next argument)
- If the displayed variable will need fewer characters than the defined width, space characters are added from left to right, and the value is right-justified
- If the displayed value needs more characters, the field width will automatically expand as needed

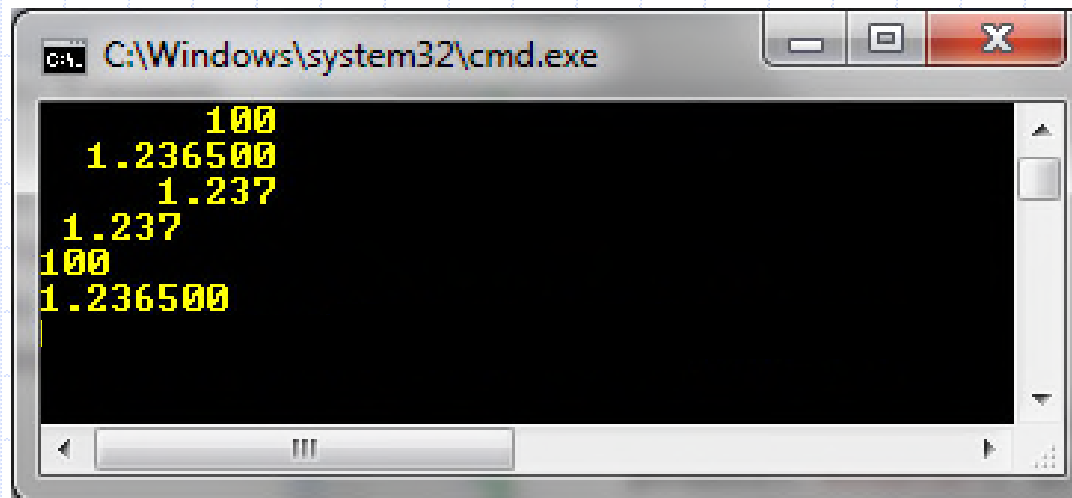


In the case of floating-point numbers, the defined width should take into account the precision digits and the decimal point

Example

```
#include <stdio.h>
int main()
{
    int a = 100;
    float b = 1.2365;

    printf("%10d\n", a);
    printf("%10f\n", b);
    printf("%10.3f\n", b);
    printf("%*.3f\n", 6, b);
    printf("%2d\n", a);
    printf("%6f\n", b);
    return 0;
}
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays the output of the C program in yellow text on a black background. The output consists of six lines, each corresponding to a printf statement in the code above. The first line shows "100" with 10 spaces before it. The second line shows "1.236500" with 10 spaces before it. The third line shows "1.237" with 10 spaces before it. The fourth line shows "1.237" with 6 spaces before it. The fifth line shows "100" with 2 spaces before it. The sixth line shows "1.236500" with 6 spaces before it.

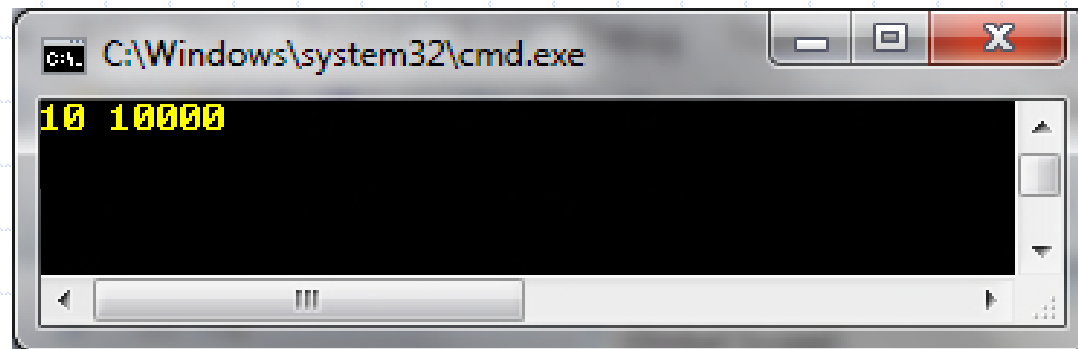
```
C:\Windows\system32\cmd.exe
100
1.236500
1.237
1.237
100
1.236500
```

Prefix

- To indicate that the displayed value is a **short** integer we can use the letter **h**
- Similarly, to indicate that the displayed value is a **long** integer we can use the letter **l**

```
#include <stdio.h>
int main()
{
    short a = 10;
    long b = 10000;

    printf("%hd %ld\n", a, b);
    return 0;
}
```



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt displays the output of the C program: '10 10000' in yellow text on a black background. The window has standard Windows controls (minimize, maximize, close) in the top right corner.

Flags

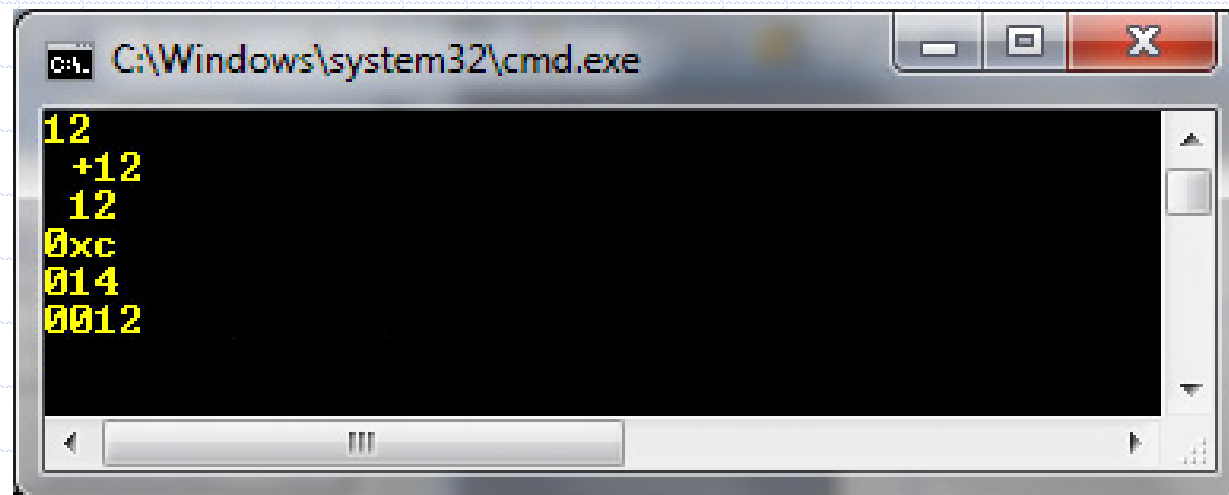
- Flags can be used to control the output of numeric values as listed in the following table

Flag	Meaning
-	Left aligns the output value within the defined field width.
+	Prefixes the output positive values with +.
space	Prefixes the output positive values with a space character.
#	Prefixes octal numbers with 0 and hex numbers with 0x or 0X. When used with floating point numbers it forces output to contain a decimal point.
0	Pads with zeros until the defined width is reached.

Example

```
#include <stdio.h>
int main()
{
    int a = 12;

    printf("%-4d\n", a);
    printf("%+4d\n", a);
    printf("% d\n", a);
    printf("%#0x\n", a);
    printf("%#o\n", a);
    printf("%04d\n", a);
    return 0;
}
```

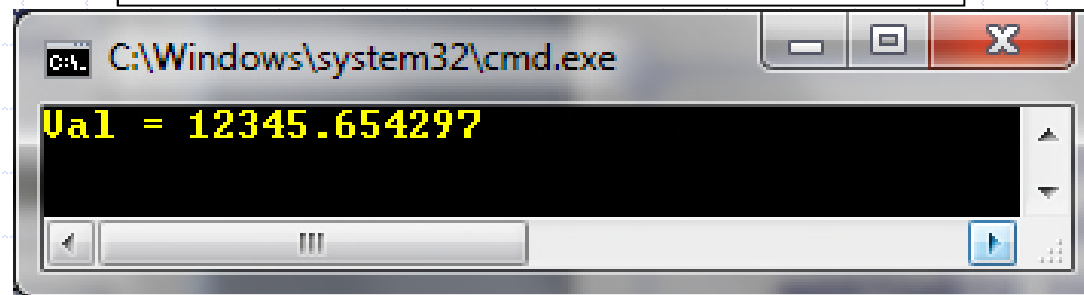


A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background and yellow text. It displays the output of the C program shown in the code block above. The output consists of six lines: "12", "+12", "12", "0xc", "014", and "0012". The first line is "12", the second is "+12", the third is "12", the fourth is "0xc", the fifth is "014", and the sixth is "0012".

Remarks (1/2)

- If you use a floating-point number variable in several statements (e.g. in comparisons, arithmetical operations, etc) prefer the `double` type rather than the `float` one, because of the unexpected manner that `float` type manages the decimal digits
- E.g. in the next program the output might not be the expected value (i.e., 12345.65432), but a similar one

```
#include <stdio.h>
int main()
{
    float a;
    a = 12345.65432;
    printf("Val = %f\n",a);
    return 0;
}
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The output displayed is "Val = 12345.654297" in yellow text on a black background. The window has standard Windows XP-style window controls (minimize, maximize, close) in the title bar.

Remarks (2/2)

- To expand the format string of `printf()` to several lines (typically for purposes of readability) use a backslash (\)
- E.g., the following `printf()` is written over three lines

```
printf("This printf uses three lines, but the \  
message will appear \  
on one line ");
```

but the output will appear on one line

Type casting

- C allows the programmer to convert (temporary) the type of an expression to another type
- This kind of conversion is known as **type casting**
- Cast expressions have the following form:

`(data_type) expression`

- `data_type` specifies the type to which the expression should be converted
- E.g. after the following declaration:

```
float a, b = 2.34;
```

the cast expression:

```
a = (int)b;
```

converts the value of `b` from `float` to `int` and make `a` equal to 2

- After being used in the cast expression, `b` will be treated again as `float`

Example

```
#include <stdio.h>
int main()
{
    int i = 20, j = 30;
    float k;

    k = (float)i/j;

    printf("%.2f\n", k);
    printf("%d\n", i);
    return 0;
}
```

- ♦ The cast expression `(float)i` converts the value of `i` from `int` to `float` and the result of the division is a decimal number
- ♦ If we had written `k = i/j`, then `k` would have been equal to 0 (i.e., the result of the integer division 20/30)
- ♦ As already noted, the conversion of `i` from `int` to `float` is temporary, so `i` remains an `int` for the rest of the program
- ♦ This is why we use `%d` (not `%f`) to display its value in the second `printf()`

