To:
Tenouk

C LAB WORKSHEET 4_1
C main() and printf() functions 2

Items in this page:

The _set_printf_count_output()

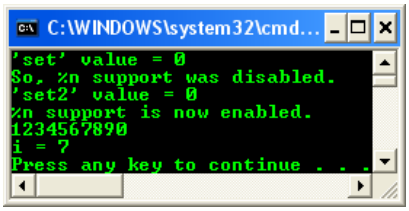
This function enable or disable support of the %n format in printf(), _printf_l(), wprintf(), _wprintf_l() family functions.

Item	Description
Function	_set_printf_count_output().
Use	This function enable or disable support of the %n format in printf(), _printf_l(), wprintf(), _wprintf_l()
Prototype	int _set_printf_count_output(int enable);
Example	see below example.
Parameters	enable - A non-zero value to enable %n support, 0 to disable %n support.
Return value	The state of %n support before calling this function: non-zero if %n support was enabled, 0 if it was disabled.
Include file	<stdio.h>
Remark	Because of security reasons, support for the %n format specifier is disabled by default in printf() and all its variants. If %n is encountered in a printf() format specification, the default behavior is to invoke the invalid parameter handler. Calling _set_printf_count_output() with a non-zero argument will cause printf-family functions to interpret %n as described in printf() Type Field Characters topic.

Table 7.

Program Example:

```
// _set_printf_count_output() and _get_printf_count_output() example
#include <stdio.h>

int main()
{
  int set, set2;
  int i;
  set = _set_printf_count_output(1);
  printf("\'set\' value = %d\n", set);
  printf("So, %%n support was %sabled.\n", set ? "en" : "dis");
  // _get_printf_count_output() returns non-zero if %n is supported,
  // 0 if %n is not supported.
  set2 = _get_printf_count_output();
  printf("\'set2\' value = %d\n", set);
  printf("%%n support is now %sabled.\n", set2 ? "en" : "dis");
  // %n format should set i to 7
  printf("1234567%n890\n", &i);
  printf("i = %d\n", i);
  return 0;
}
```



The Width Specification

The second optional field of the format specification is the width specification. The width argument is a nonnegative decimal integer controlling the minimum number of characters printed. If the number of characters in the output value is less than the specified width, blanks are added to the left or the right of the values - depending on whether the - flag (for left alignment) is specified - until the minimum width is reached. If width is prefixed with 0, zeros are added until the minimum width is reached (not useful for left-aligned numbers).
The width specification never causes a value to be truncated. If the number of characters in the output value is greater than the specified width, or if width is not given, all characters of the value are printed.
If the width specification is an asterisk (*), an int argument from the argument list supplies the value. The width argument must precede the value being formatted in the argument list. A nonexistent or small field width does not cause the truncation of a field; if the result of a conversion is wider than the field width, the field expands to contain the conversion result.

The Precision Specification

The third optional field of the format specification is the precision specification. It specifies a nonnegative decimal integer, preceded by a period (.), which specifies the number of characters to be printed, the number of decimal places, or the number of significant digits. Unlike the width specification, the precision specification can cause either truncation of the output value or rounding of a floating-point value. If precision is specified as 0 and the value to be converted is 0, the result is no characters output, as shown below:

```
printf("%.0d", 0);  /* No characters output */
```

If the precision specification is an asterisk (*), an int argument from the argument list supplies the value. The precision argument must precede the value being formatted in the argument list. The type determines the interpretation of precision and the default when precision is omitted, as shown in the following table.

How Precision Values Affect Type

The following Table lists types and precisions.

Type	Meaning	Default
a, A	The precision specifies the number of digits after the point.	Default precision is 6. If precision is 0, no point is printed unless the # flag is used.
c, C	The precision has no effect.	Character is printed.
d, i, u, o, x, X	The precision specifies the minimum number of digits to be printed. If the number of digits in the argument is less than precision, the output value is padded on the left with zeros. The value is not truncated when the number of digits exceeds precision.	Default precision is 1.
e, E	The precision specifies the number of digits to be printed after the decimal point. The last printed digit is rounded.	Default precision is 6; if precision is 0 or the period (.) appears without a number following it, no decimal point is printed.
f	The precision value specifies the number of digits after the decimal point. If a decimal point appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.	Default precision is 6; if precision is 0, or if the period (.) appears without a number following it, no decimal point is printed.
g, G	The precision specifies the maximum number of significant digits printed.	Six significant digits are printed, with any trailing zeros truncated.
s, S	The precision specifies the maximum number of characters to be printed. Characters in excess of precision are not printed.	Characters are printed until a null character is encountered.

Table 8.

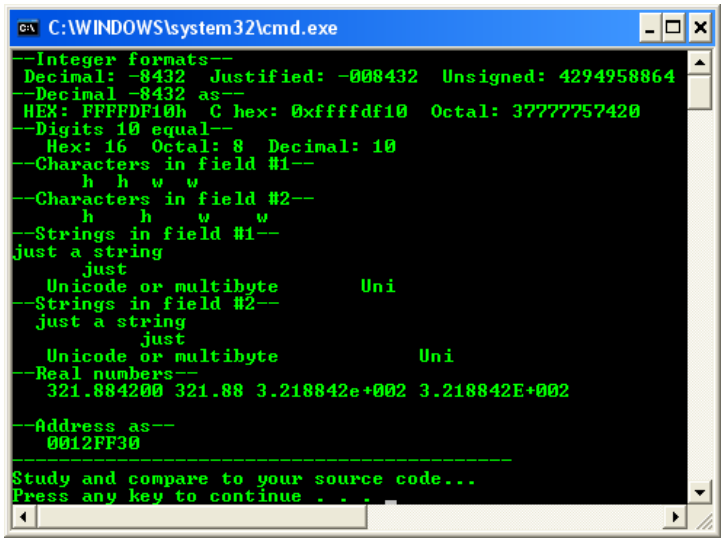
If the argument corresponding to a floating-point specifier is infinite, indefinite, or NAN (**N**ot **A** **N**umber - .Net framework), printf() gives the following output.

Value	Output
+ infinity	1.#INFrandom-digits (INF – Infinite number)
– infinity	–1.#INFrandom-digits
Indefinite (same as quiet NaN – **N**ot **a** **N**umber)	digit.#INDrandom-digits (IND – Indefinite number)
NAN	digit.#NANrandom-digits

Table 9.

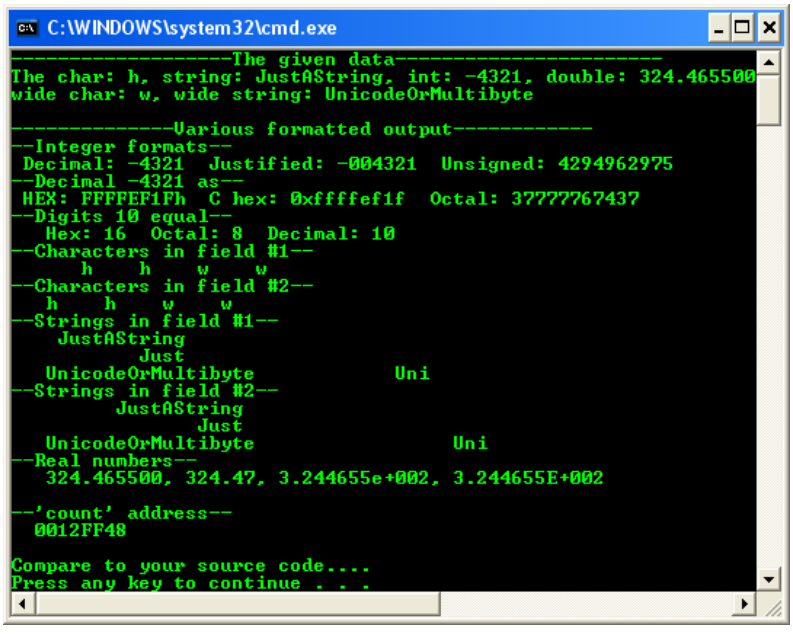
Program Example:

```
// using the printf() and wprintf() functions to produce formatted output example.
#include <stdio.h>

int main(void)
{
   char     chs = 'h', *string = "just a string";
   wchar_t  wch = L'w', *wstring = L"Unicode or multibyte";
   int      count = -8432;
   double   fpnt = 321.8842;
   // display integers
   printf("--Integer formats--\n Decimal: %d  Justified: %.6d  "
          "Unsigned: %u\n", count, count, count, count);
   // display decimals
   printf( "--Decimal %d as--\n HEX: %Xh  C hex: 0x%x  Octal: %o\n", count, count, count, count);
   // display in different radixes
   printf( "--Digits 10 equal--\n   Hex: %i  Octal: %i  Decimal: %i\n", 0x10, 010, 10);
   // display characters
   printf("--Characters in field #1--\n"%7c%3hc%3C%3lc\n", chs, chs, wch, wch);
   wprintf(L"--Characters in field #2--\n"
          L"%7C%5hc%5c%5lc\n", chs, chs, wch, wch);
   // display strings
   printf("--Strings in field #1--\n%10s\n"
         "%10.4hs\n   %S%10.3ls\n",string, string, wstring, wstring);
   wprintf(L"--Strings in field #2--\n%15S\n"
          L"%15.4hs\n   %s%15.3ls\n",string, string, wstring, wstring);
   // display real numbers
   printf("--Real numbers--\n   %f %.2f %e %E\n",fpnt, fpnt, fpnt, fpnt);
   // display pointer
   printf("\n--Address as--\n   %p\n", &count);
   printf("-----------------------------------------\n");
   printf("Study and compare to your source code...\n");
   return 0;
}
```



Program Example:

```c
/* This program uses the printf_s() and wprintf_s() functions
 * to produce formatted output. */
#include <stdio.h>

int main(void)
{
   char   chs = 'h', *string = "JustAString";
   int    count = -4321;
   double fpnt = 324.4655;
   wchar_t wch = L'w', *wstring = L"UnicodeOrMultibyte";
   printf("-------------------The given data-----------------------\n");
   printf("The char: %c, string: %s, int: %d, double: %f\n"
        "wide char: %lc, wide string: %ls\n", chs, string, count, fpnt, wch, wstring);
   printf("\n--------------Various formatted output------------\n");
   /* display integers. */
   printf_s("--Integer formats--\n"
          " Decimal: %d  Justified: %.6d  Unsigned: %u\n",count, count, count, count );
   printf_s("--Decimal %d as--\n HEX: %Xh  C hex: 0x%x  Octal: %o\n",count, count, count, count );
   /* display in different radixes. */
   printf_s("--Digits 10 equal--\n   Hex: %i  Octal: %i  Decimal: %i\n",0x10, 010, 10 );
   /* display characters. */
   printf_s("--Characters in field #1--\n%7c%5hc%5C%5lc\n", chs, chs, wch, wch);
   wprintf_s(L"--Characters in field #2--\n%4C%5hc%5c%5lc\n", chs, chs, wch, wch);
   /* display strings. */
   printf_s("--Strings in field #1--\n%15s\n%15.4hs\n   %S%15.3ls\n",string, string, wstring, wstring);
   wprintf_s(L"--Strings in field #2--\n%20S\n%20.4hs\n   %s%20.3ls\n",string, string, wstring, wstring);
   /* display real numbers. */
   printf_s("--Real numbers--\n   %f, %.2f, %e, %E\n", fpnt, fpnt, fpnt, fpnt);
   /* display pointer. */
   printf_s("\n--\'count\' address--\n  %p\n", &count);
   printf("\nCompare to your source code....\n");
     return 0;
}
```



Question:

From the previous program example, study and find codes that you don't understand. Then discuss with your class member and create five questions with answers based on the codes that you don't understand.

More Story...

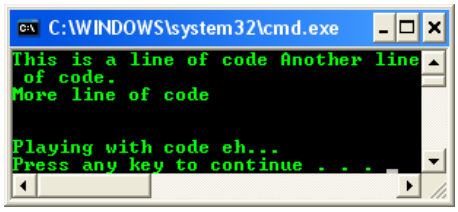
The printf() require an argument to be passed to it. It is a format string argument for example:

The code: printf("My name is Mr. C. Cplusplus\n");

The output: My name is Mr. C. Cplusplus

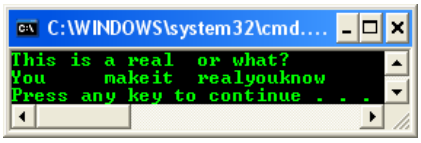
The format string is the string of characters enclosed between the double quotes, "", inside the pair of parentheses, ( ). The items inside the parentheses are used to pass arguments.
Delete all your previous code in the main() body. Then, try the following code snippet and see what the \n does.

```
printf("This is a line of code ");
printf("Another line\n of code. \nMore line of code \n");
printf("\n\nPlaying with code eh...\n");
```



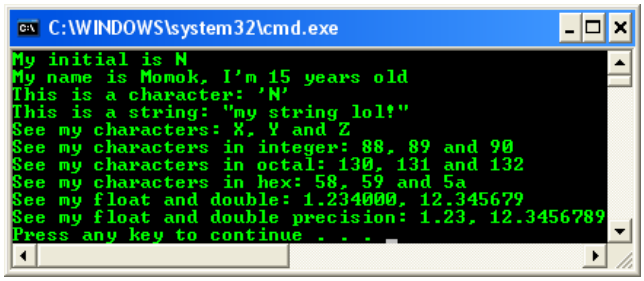
Try other escape sequence, \t code by replacing the previous code with the following code.

```
printf("This is\ta real\tor what?\n");
printf("You\tmake\it\treal\you\know\n");
```



Next, try the following conversion specifier code. Study the output and compare it with the related source code.

```
char chr = 'N';

printf("My initial is %c\n", chr);
printf("My name is %s, I\'m %d years old\n", "Momok", 15);
printf("This is a character: \'%c\'\n", chr);
printf("This is a string: \"%s\"\n", "my string lol!");
printf("See my characters: %c, %c and %c\n", 'X', 'Y', 'Z');
printf("See my characters in integer: %d, %d and %d\n", 'X', 'Y', 'Z');
printf("See my characters in octal: %o, %o and %o\n", 'X', 'Y', 'Z');
printf("See my characters in hex: %0X, %0X and %0x\n", 'X', 'Y', 'Z');
printf("See my float and double: %f, %f\n", 1.234, 12.34567890);
printf("See my float and double precision: %.2f, %.7f\n", 1.234, 12.34567890);
```



Question:

From the previous program example, study and find codes that you don't understand. Then discuss with your class member and create three questions with answers based on the codes that you don't understand.

Questions And Activities:

1. How many '\n's are needed to escape one blank line? **Ans:** 1
2. What is the effect of the '\t' escape character? **Ans:** provide 1 horizontal tab
3. What was printed in place of %s format specifier? **Ans:** a string
4. What was printed in place of %c format specifier? **Ans:** a character
5. What was printed in place of %d format specifier? **Ans:** a decimal
6. What was printed in place of %f format specifier? **Ans:** a floating point number with default 6 precision
7. What %.3f means? **Ans:** a floating point number with 3 precision
8. From the code sample you can see that the decimal, octal, hexadecimal and character can be converted each other by using a proper format specifier. What are the format specifiers for octal

and hexadecimal representatives? **Ans:** %o for octal and %x or %X for hexadecimal

9. Shows the output for the following codes.

a. printf("Get me %d \n tickets \n ", 10);
printf("\n\n****\n ");

```
Get me 10
 tickets


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

b. printf("Qty = %d \n ", 10);
printf("Price = %.2f\n Total = %.2f\n", 2.25, 22.50);

```
Qty = 10
 Price = 2.25
 Total = 22.50
Press any key to continue . . .
```

c. printf("****\t******\t\n ");
printf("\n***\t**\n ");

```
****    ******

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

d. printf("***%c***\n ", 'W');
printf("%s***\n", "TEST ");

```
***W***
 TEST ***
Press any key to continue . . .
```

10. Write a complete program for the following output.

a. Pass "handsome", 20 and 70.45 as arguments.

I am handsome, 20 years old and my weight is 70.45 kg.

```
#include <stdio.h>

int main()
{
    char appearance[20];
    int age = 0;
    float weight = 0;
    printf("Pass a string, an integer and a float:\n");
    // scanf_s("%s %d %f", appearance, 19, &age, &weight);
    //  a secure version
    scanf("%s %d %f", appearance, &age, &weight);
    printf("I am %s, %d years old and my weight is %.2f kg.\n",
                                    appearance, age, weight);
    return 0;
}
```

```
Pass a string, an integer and a float:
Handsome 20 70.45
I am Handsome, 20 years old and my weight is 70.45 kg.
Press any key to continue . . .
```

b. Pass 'R', 321 and "Online" as argument.

The letter is R
321 is a whole number
Online is a string

```
#include <stdio.h>

int main()
{
   char astring[20];
   int whnum = 0;
   char achar = ' '; // a space as an initial character value
   // a secure version
   // scanf_s("%c %d %s", &achar, 1, &whnum, astring, 19);
   scanf("%c %d %s", &achar, &whnum, astring);
   printf("The letter is %c\n%d is a whole number\n%s is a string\n",
                                achar, whnum, astring);
   return 0;
}
```

```
R 321 Online
The letter is R
321 is a whole number
Online is a string
Press any key to continue . . .
```

11. Write a complete C program that contains the following components.

a. printf().
b. 3 escape characters.
c. 4 format specifiers.

```
#include <stdio.h>

int main()
{
   int anum = 70;
   char achar = 'A';
   double adouble = 1.234;
   printf("A character: \"%c\",\tan integer: %d,\na"
       " floating point number: %.4f and a hex: %0X.\n", achar, anum,
                                       adouble, anum);
return 0;
}
```

```
A character: "A",       an integer: 70,
a floating point number: 1.2340 and a hex: 46.
Press any key to continue . . .
```

12. What wrong with the following codes if any, and how to correct it?

1.
```
#include <stdio.h>

int main()
{
    printf("My first C class is very terrible!\n ");
}
```

No return an integer such as return 0; statement. The main() return value is an integer as in int main(), so main() expect an integer to be returned.

2.
```
#include <stdio.h>

void main()
{
    printf("My first C class is very terrible!\n ");
    return 0;
}
```

The return 0; statement must be removed because main() doesn't have a return value denoted by the void keyword as in void main().

Supplementary Note - C Run-Time errno Constants

The #include <errno.h> preprocessor directive provides runtime errno constants. The errno values are constants assigned to errno in the event of various error conditions. errno.h contains the definitions of the errno values. However, not all the definitions given in errno.h are used in 32-bit Windows operating systems. Some of the values in errno.h are present to maintain compatibility with the UNIX family of operating systems. The errno values in a 32-bit Windows operating system are a subset of the values for errno in XENIX systems. Thus, the errno value is not necessarily the same as the actual error code returned by a system call from the Windows operating systems. To access the actual operating system error code, use the _doserrno variable, which contains this value. The following errno values are supported:

errno value	Description
ECHILD	No spawned processes.
EAGAIN	No more processes. An attempt to create a new process failed because there are no more process slots, or there is not enough memory, or the maximum nesting level has been reached.
E2BIG	Argument list too long.
EACCES	Permission denied. The file's permission setting does not allow the specified access. This error signifies that an attempt was made to access a file (or, in some cases, a directory) in a way that is incompatible with the file's attributes. For example, the error can occur when an attempt is made to read from a file that is not open, to open an existing read-only file for writing, or to open a directory instead of a file. Under MS-DOS operating system versions 3.0 and later, EACCES may also indicate a locking or sharing violation. The error can also occur in an attempt to rename a file or directory or to remove an existing directory.
EBADF	Bad file number. There are two possible causes: 1. The specified file descriptor is not a valid value or does not refer to an open file. 2. An attempt was made to write to a file or device opened for read-only access.
EDEADLOCK	Resource deadlock would occur. The argument to a math function is not in the domain of the function.
EDOM	Math argument.
EEXIST	Files exist. An attempt has been made to create a file that already exists. For example, the _O_CREAT and _O_EXCL flags are specified in an _open() call, but the named file already exists.
EILSEQ	Illegal sequence of bytes (for example, in an MBCS string).
EINVAL	Invalid argument. An invalid value was given for one of the arguments to a function. For example, the value given for the origin when positioning a file pointer (by means of a call to fseek()) is before the beginning of the file.
EMFILE	Too many open files. No more file descriptors are available, so no more files can be opened.
ENOENT	No such file or directory. The specified file or directory does not exist or cannot be found. This message can occur whenever a specified file does not exist or a component of a path does not specify an existing directory.
ENOEXEC	Exec (e.g.: _exec()) format error. An attempt was made to execute a file that is not executable or that has an invalid executable-file format.
ENOMEM	Not enough core. Not enough memory is available for the attempted operator. For example, this message can occur when insufficient memory is available to execute a child process, or when the allocation request in a _getcwd() call cannot be satisfied.
ENOSPC	No space left on device. No more space for writing is available on the device (for example, when the disk is full).
ERANGE	Result too large. An argument to a math function is too large, resulting in partial or total loss of significance in the result. This error can also occur in other functions when an argument is larger than expected (for example, when the buffer argument to _getcwd is longer than expected).
EXDEV	Cross-device link. An attempt was made to move a file to a different device (using the rename() function).
STRUNCATE	A string copy or concatenation resulted in a truncated string.

Table 10

Finally, why you need to learn this Module? **Ans:** Just for fun!!! (this is one of the correct answer!)

The C main() and printf() Functions: Part 1 | Part 2