To:
Tenouk

C LAB WORKSHEET 4
C main() and printf() functions 1

Items in this page:

1. Be familiar with the compiler – more on project options.
2. main() function – the need of main() as C/C++ execution point.
3. Tutorial references are: C/C++ intro & brief history, C/C++ data type 1, C/C++ data type 2, C/C++ data type 3 and C/C++ statement, expression & operator 1, C/C++ statement, expression & operator 2 and C/C++ statement, expression & operator 2. More printf() and its family examples can be found in C formatted input/output. A complete story of main() is in C and C++ main() story.

If you have gone through Module 1 and 2, the main() function is needed for the execution point of C/C++ programs. That means compiler start the program execution at main() function. A complete story about the main() function is given in Module Y. In your C/C++ program also, there are other functions that you already familiar with such as printf(). You can see then, C/C++ programs just consist of functions with main() as the execution point.

1. Create a new empty Win32 console application project named **mymain** and add a C++ source file named **mymainsrc** as done in previous lab practice. Make sure you set this project to be compiled as C code. Type the following codes.

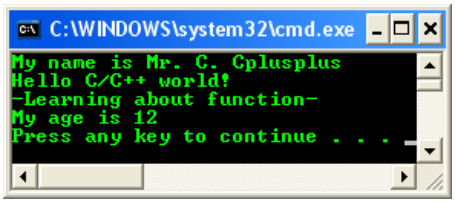
```
#include <stdio.h>

void main(void)
{ }
```

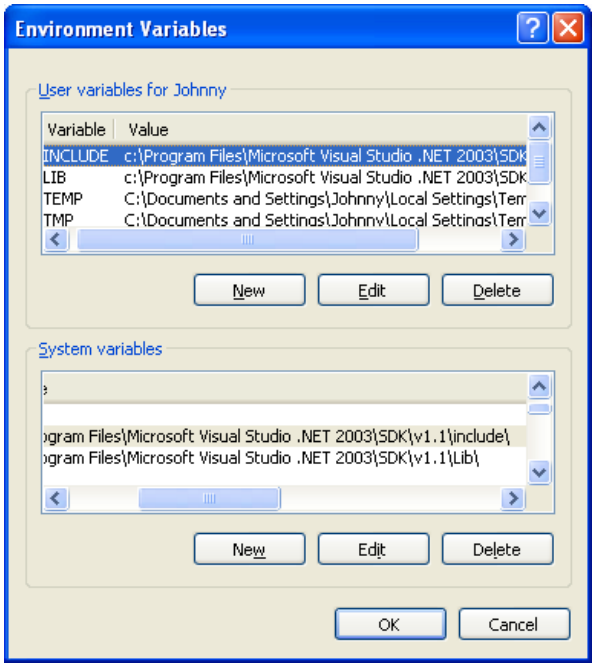
2. Then add more codes. You can omit the comments.

```
#include <stdio.h>
// this '//' means line comment...will be ignored by compiler
// the first void means main() doesn't return anything
// the second void means, main() doesn't receive anything
void main(void)
// the beginning of the main/other function body is indicated by } – a curly brace
{
    // declare and initialize a variable with integer type...
    // every C/C++ statement will be terminated by ; (semi colon)
    int MyAge = 12;
    // printf() is another built-in/standard function for standard
    // output - your terminal/console/screen that defined in stdio.h.
    // In the following three printf()s, they receive and return
    // formatted strings...
    // the \n is an escape to new line...more on this later...
    printf("My name is Mr. C. Cplusplus\n");
    printf("Hello C/C++ world!\n");
    printf("-Learning about function-\n");
    // the following printf() return the age...
    printf("My age is %d\n", MyAge);
    // no return statement coz main() return nothing (void)
    // the end of the main()/other function's body is indicated by }
}
```

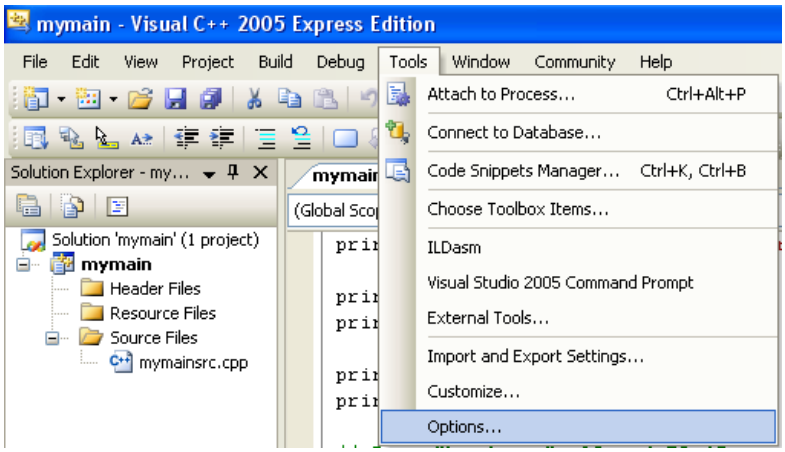
3. Build and run your program. The output should be as shown.



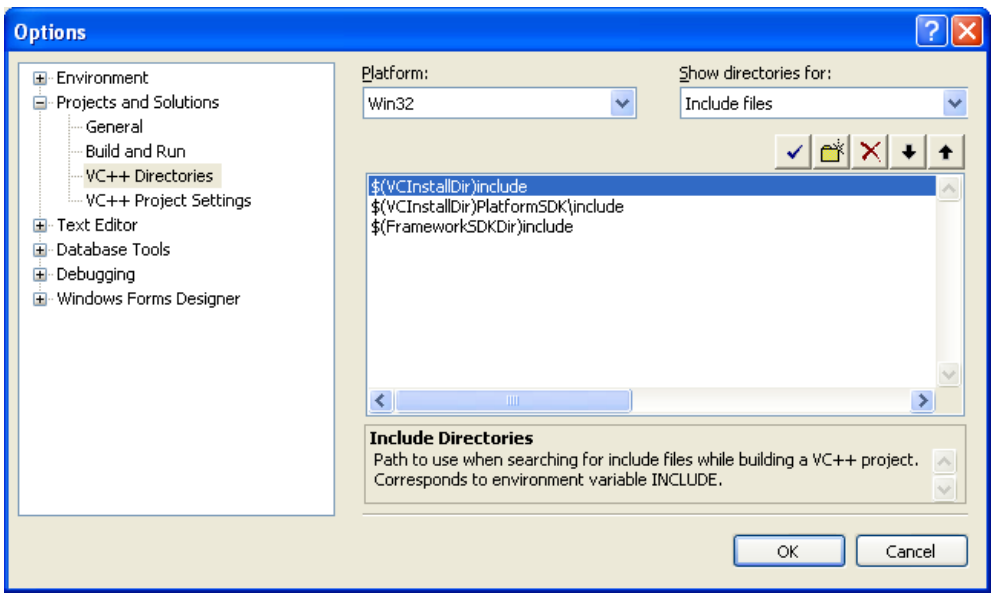
4. In the program we define the main() function but what about printf()? The definition of the printf() is in **stdio.h** header file. That is why we need to include #include <stdio.h> in our program so that compiler know what the printf() is. We pass some string argument to printf() function and then it return the string to the screen – the standard output.
5. The #include is called **preprocessor directive**. In a simple word, we instruct a compiler to find stdio.h file and read/process it.
6. Why we do this? In the C/C++ programs, for commonly and frequently used routines such as printf(), no need for us to retype the code for printf() function's definition, how it works, declaration etc., just include in our program using the #include preprocessor directive and it make our program smaller and structured or modular. Don't forget also the printf() and other standard functions are reusable.
7. By using the #include directive, compiler know what file to find but how your compiler find these include files? During the installation, the include and library paths have been set in your environment variables. For Windows XP Pro example is shown on the right.



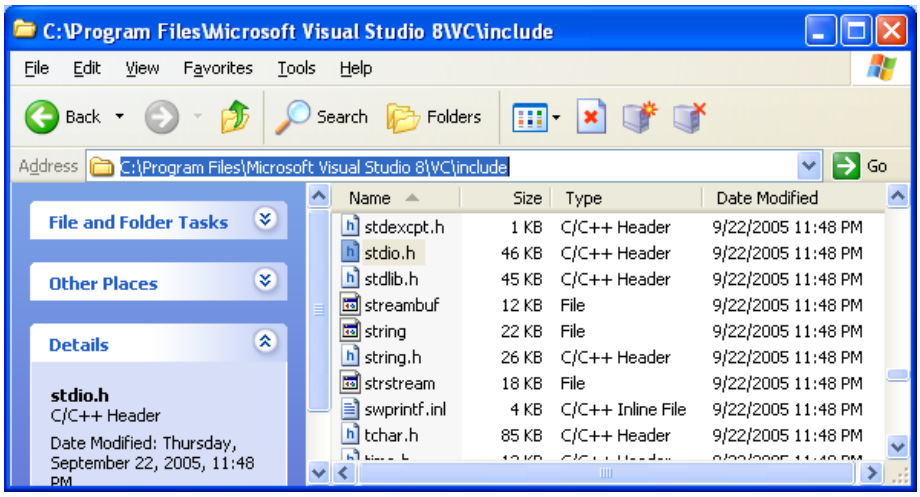
8. Then, how your compiler search those include (and other project related files) files if there are more than one include paths (for example if your machine has more than one version of the VC++)? The paths and sequence have been set in your project **Options**. Click **Tools** → **Options** menu.

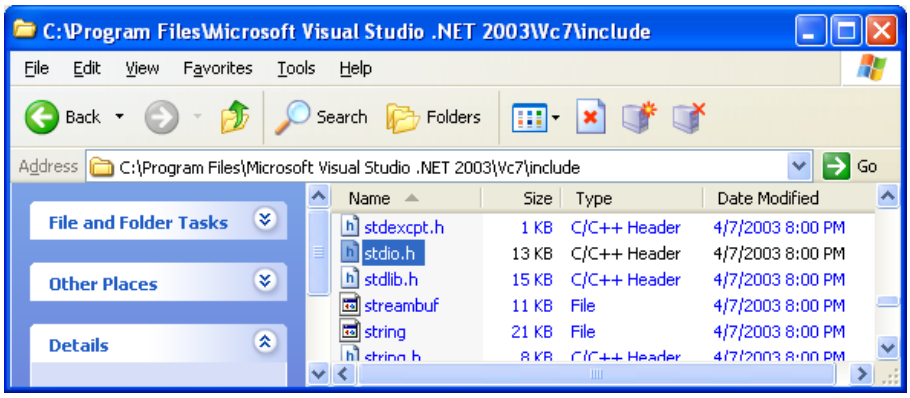


9. Expand the **Projects and Solutions** folder and select **VC++ Directories** link. In the **Show directories for:** select **Include files**. The paths are given there. You can add (if you have third party, non standard or user-defined include files) or remove the not useable include files. Don't forget to explore other options in the **Options** form but do not change any setting.

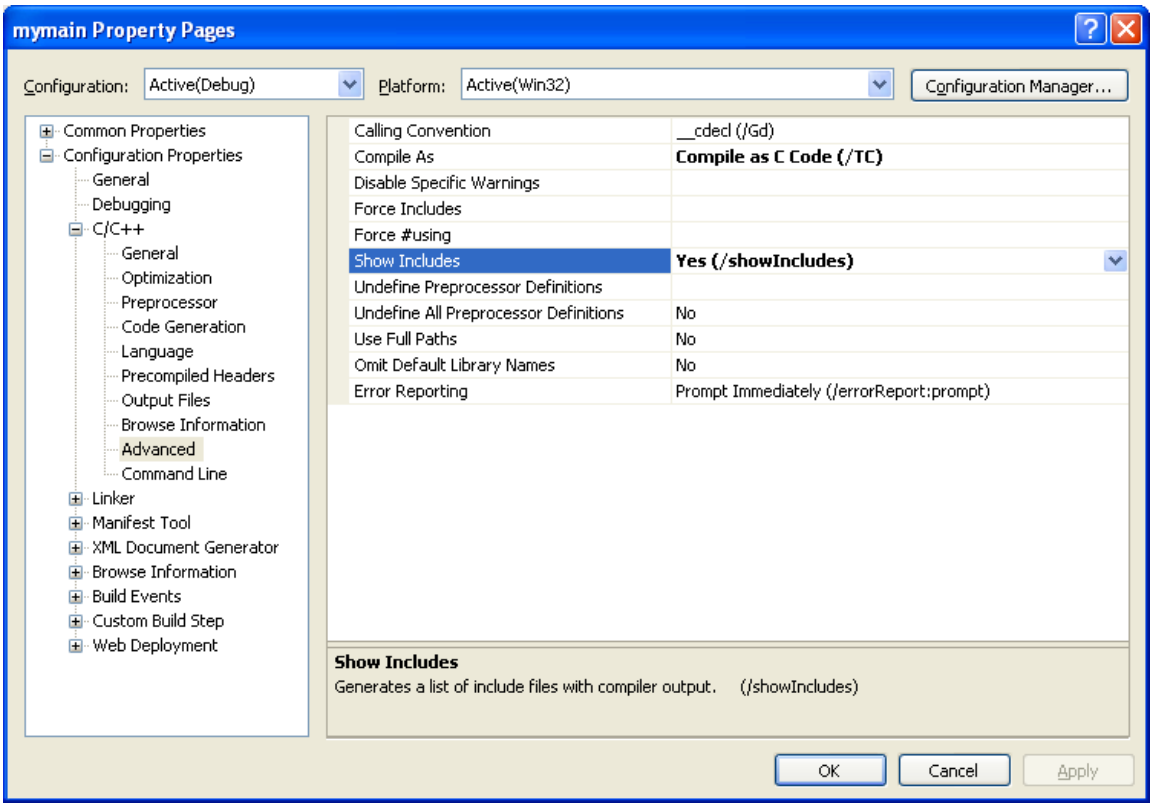


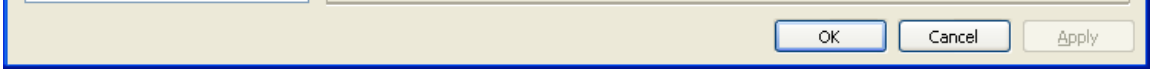
10. Finally, you can check the location/path of the include files physically and for stdio.h example is shown below for Visual C++ 2005 EE (version 8) and Visual Studio .Net 2003 (version 7) that were installed on Windows XP Pro.



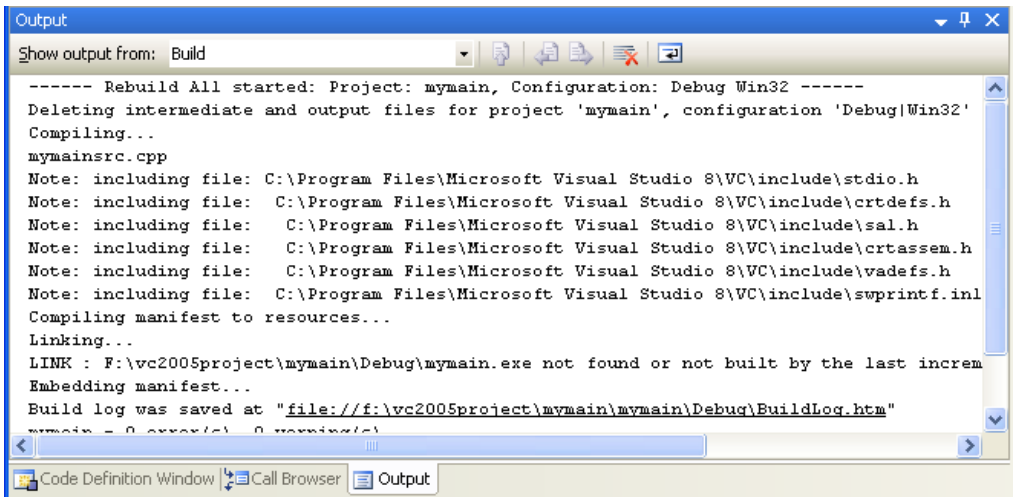


11. You can also see that under the include folder, there are also header files without the .h extensions. Those header files used for C++ programs.
12. The file must be found and read before any function it defined can be used in our program including the main() and that is why it is put outside the main() body, at the beginning of the program. It is in global space so that any function that stdio.h defined such as printf() can be used anywhere in the program/file/source code or other file/source code that the main() program calls.
13. If you want to see those include files that are used during the building process, you can set the **Show Includes** option in the project setting.
14. Click the **Project** ⟶ ***your_project_name*** **Properties** menu. Select **Advanced** link under **C/C++** sub folder. Change the **Show Includes** option to **Yes (/showincludes)** and click the **Apply** button. Close the property form and rebuild your program.

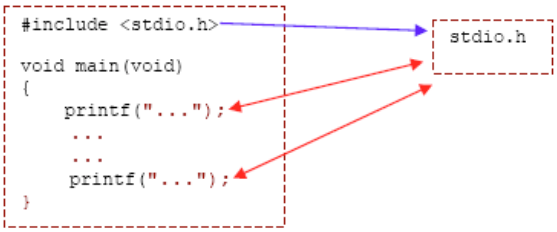




15. See messages in the **Output** window. It should be as shown below. Not just stdio.h file, there are other header files as well.



16. Back to our main discussion, so, we should agree at this moment, C/C++ program just contain functions. You will learn a lot more C/C++ standard functions that normally provided together with the compilers. This standard function collection in a compiled form normally called libraries.
17. Keep in mind that there are a lot more non-standard functions (libraries) and user-defined functions out there. Both non-standard and user-defined functions normally used to program specific tasks such as for graphic manipulation and search routines. You may also encounter other terms used for function in other programming languages such as routine and procedure. The following Figure should explain the previous example clearer.



18. We have main() function that doesn't receive any argument and doesn't pass any value, as the execution point. Then, in main() function, we call printf() function. During the function calls, we may receive and/or pass argument(s)/value(s). The one-way blue arrow means finding the stdio.h file for printf() definition and the two-way red arrows means the printf() function may pass and/or receive argument(s) to be processed.

Questions: Please answer the following questions:

1. Name the function that must always exist as an execution point in C/C++ program. **Ans:** main()
2. Why the stdio.h header file need to be at the beginning of the source code file? **Ans:** So that any functions defined in stdio.h header file such as printf() and scanf() can be used immediately in the program after the stdio.h line of code.
3. What indicates there is no arguments are being received by main()? **Ans:** the void keyword as in main(void).
4. What indicates there is no values are being returned by main()? **Ans:** the void keyword as in void main().
5. What indicates the beginning and the end of the main() body? **Ans:** The opening and closing curly braces, { and } for the beginning and the end of the main() body respectively.
6. What is used to terminate each C/C++ statement? **Ans:** a semicolon, ;.
7. Where is the printf() function defined? **Ans:** In stdio.h header file.
8. How can we tell compiler to find the printf() definition? **Ans:** By including the stdio.h header file in our program before the printf() function been used.

The printf(), printf_s(), _printf_s_l(), wprintf_s(), _wprintf_s_l() Family Story

The definitions are summarized in the following Tables.

Item	Description
Function	printf() family.
Use	These functions family print formatted output to the standard output stream. The secure versions are printf_s(), _printf_s_l(), wprintf_s() and _wprintf_s_l().
Prototype	1. int printf(const char *format [, argument]...); 2. int _printf_l(const char *format, locale_t locale [, argument]...); 3. int wprintf(const wchar_t *format [, argument]...); 4. int _wprintf_l(const wchar_t *format, locale_t locale [, argument]...);
Example	printf("See my characters: %c, %c and %c\n", 'X', 'Y', 'Z');

Parameters	format - Format control. argument - Optional arguments. locale - The locale to use.
Return value	Returns the number of characters printed, or a negative value if an error occurs. If format is NULL, the invalid parameter handler is invoked. If execution is allowed to continue, the function returns -1 and sets errno to EINVAL.
Include file	<stdio.h> or <wchar.h> for wprintf_s() and _wprintf_S_l().
Remark	-

Table 1

Item	Description
Function	Secure printf() family.
Use	Similar to the previous use but are secure versions of printf(), _printf_l(), wprintf(), _wprintf_l().
Prototype	1. int printf_s(const char *format [, argument]...); 2. int _printf_s_l(const char *format, locale_t locale [, argument]...); 3. int wprintf_s(const wchar_t *format [, argument]...); 4. int _wprintf_s_l(const wchar_t *format, locale_t locale [, argument]...);
Example	printf_s("Decimal: %d Justified: %.6d Unsigned: %u\n", count, count, count);
Parameters	format - Format control. argument - Optional arguments. locale - The locale to use.
Return value	Returns the number of characters printed, or a negative value if an error occurs. If format is NULL, the invalid parameter handler is invoked, as described in Parameter Validation. If execution is allowed to continue, the function returns -1 and sets errno to EINVAL.
Include file	<stdio.h> or <wchar.h> for wprintf() and _wprintf_l().
Remark	The printf_s() function formats and prints a series of characters and values to the standard output stream, stdout. If arguments follow the format string, the format string must contain specifications that determine the output format for the arguments. The **main difference** between printf_s() and printf() is that printf_s() checks the format string for valid formatting characters, whereas printf() only checks if the format string is a null pointer. If either check fails, an invalid parameter handler is invoked. If execution is allowed to continue, the function returns -1 and sets errno to EINVAL. printf_s() and fprintf_s() behave identically except that printf_s() writes output to stdout rather than to a destination of type FILE. wprintf_s() is a wide-character version of printf_s(); format is a wide-character string. wprintf_s() and printf_s() behave identically if the stream is opened in ANSI mode. printf_s() doesn't currently support output into a UNICODE stream. The versions of these functions with the _l suffix are identical except that they use the locale parameter passed in instead of the current thread locale. The format argument consists of ordinary characters, escape sequences, and (if arguments follow format) format specifications. The ordinary characters and escape sequences are copied to stdout in order of their appearance. For example, the line: printf("Line one\n\t\tLine two\n"); produces the output: Line one Line two

Table 2

The Format Specifications

Format specifications always begin with a percent sign (%) and are read left to right. When printf() encounters the first format specification (if any), it converts the value of the first argument after format and outputs it accordingly. The second format specification causes the second argument to be converted and output, and so on. If there are more arguments than there are format specifications, the extra arguments are ignored. The results are undefined if there are not enough arguments for all the format specifications.
This topic describes the syntax for format specifications fields, used in printf(), wprintf() and their related families. A format specification, which consists of optional and required fields, has the following form:

```
%[flags] [width] [.precision] [{h | l | ll | I | I32 | I64}]type
```

Each field of the format specification is a single character or a number signifying a particular format option. The simplest format specification that you normally use contains only the percent sign and a type character (for example, %s). If a percent sign is followed by a character that has no meaning as a format field, the character is copied to stdout. For example, to print a percent-sign character, use %%. The optional fields, which appear before the type character, control other aspects of the formatting, as follows:

Parameter	Description
type	Required character that determines whether the associated argument is interpreted as a character, a string, or a number.
flags	Optional character or characters that control justification of output and printing of signs, blanks, decimal points, and octal and hexadecimal prefixes. More than one flag can appear in a format specification.
width	Optional number that specifies the minimum number of characters.
precision	Optional number that specifies the maximum number of characters printed for all or part of the output field, or the minimum number of digits printed for integer values.

H | l | ll | I | I32 | I64	Optional prefixes to type-that specify the size of argument.

Table 3

The Optional Prefixes to Type

The optional prefixes to type, h, l, I, I32, I64, and ll specify the "size" of argument (long or short, 32- or 64-bit, single-byte character or wide character, depending upon the type specifier that they modify). These type-specifier prefixes are used with type characters in printf() functions or wprintf () functions to specify interpretation of arguments, as shown in the following table. These prefixes are Microsoft extensions and are not ANSI-compatible. The h and l prefixes are Microsoft extensions when used with data of type char and you won't found it in other compiler.

The Size Prefixes for printf() and wprintf() Format-Type Specifiers

The following Table is a list of size prefixes.

To specify	Use prefix	With type specifier
long int	l (lowercase L)	d, i, o, x, or X
long unsigned int	l	o, u, x, or X
long long	ll	d, i, o, x, or X
short int	h	d, i, o, x, or X
short unsigned int	h	o, u, x, or X
__int32	I32	d, i, o, x, or X
unsigned __int32	I32	o, u, x, or X
__int64	I64	d, i, o, x, or X
unsigned __int64	I64	o, u, x, or X
ptrdiff_t (that is, __int32 on 32-bit platforms, __int64 on 64-bit platforms)	I	d, i, o, x, or X
size_t (that is, unsigned __int32 on 32-bit platforms, unsigned __int64 on 64-bit platforms)	I	o, u, x, or X
long double	l or L	f
Single-byte character with printf functions	h	c or C
Single-byte character with wprintf functions	h	c or C
Wide character with printf functions	l	c or C
Wide character with wprintf functions	l	c or C
Single-byte – character string with printf functions	h	s or S
Single-byte – character string with wprintf functions	h	s or S
Wide-character string with printf functions	l	s or S
Wide-character string with wprintf functions	l	s or S
Wide character	w	c
Wide-character string	w	s

Table 4.

Thus to print single-byte or wide-characters with printf() functions and wprintf() functions, use format specifiers as follows.

To print character as	Use function	With format specifier
single byte	printf()	c, hc, or hC
single byte	wprintf()	C, hc, or hC
wide	wprintf()	c, lc, lC, or wc
wide	printf()	C, lc, lC, or wc

Table 5.

To print strings with printf() functions and wprintf() functions, use the prefixes h and l analogously with format type-specifiers s and S.

A Type Character

The type character is the only required format field; it appears after any optional format fields. The type character determines whether the associated argument is interpreted as a character, string, or number. The types C, n, p, and S, and the behavior of c and s with printf() functions, are Microsoft extensions and are not ANSI/ISO compatible and you might not find it in other compilers.

A printf() Type Field Characters

The following Table summarizes the type field characters for printf().

Character	Type	Output format
c	int or wint_t	When used with printf() functions, specifies a single-byte character; when used with wprintf() functions, specifies a wide character.
C	int or wint_t	When used with printf() functions, specifies a wide character; when used with wprintf() functions, specifies a single-byte character.
d	int	Signed decimal integer.
i	int	Signed decimal integer.
o	int	Unsigned octal integer.
u	int	Unsigned decimal integer.
x	int	Unsigned hexadecimal integer, using "abcdef".

X	int	Unsigned hexadecimal integer, using "ABCDEF".
e	double	Signed value having the form [–]d.dddd e [sign]dd[d] where d is a single decimal digit, dddd is one or more decimal digits, dd[d] is two or three decimal digits depending on the output format and size of the exponent, and sign is + or –.
E	double	Identical to the e format except that E rather than e introduces the exponent.
f	double	Signed value having the form [–]dddd.dddd, where dddd is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.
g	double	Signed value printed in f or e format, whichever is more compact for the given value and precision. The e format is used only when the exponent of the value is less than –4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it.
G	double	Identical to the g format, except that E, rather than e, introduces the exponent (where appropriate).
a	double	Signed hexadecimal double precision floating point value having the form [−]0xh.hhhh p±dd, where h.hhhh are the hex digits (using lower case letters) of the mantissa, and dd are one or more digits for the exponent. The precision specifies the number of digits after the point.
A	double	Signed hexadecimal double precision floating point value having the form [−]0Xh.hhhh P±dd, where h.hhhh are the hex digits (using capital letters) of the mantissa, and dd are one or more digits for the exponent. The precision specifies the number of digits after the point.
n	Pointer to integer	Number of characters successfully written so far to the stream or buffer; this value is stored in the integer whose address is given as the argument.
p	Pointer to void	Prints the address of the argument in hexadecimal digits.
s	String	When used with printf() functions, specifies a single-byte–character string; when used with wprintf () functions, specifies a wide-character string. Characters are printed up to the first null character or until the precision value is reached.
S	String	When used with printf() functions, specifies a wide-character string; when used with wprintf() functions, specifies a single-byte–character string. Characters are printed up to the first null character or until the precision value is reached.

Table 6.

If the argument corresponding to %s or %S is a null pointer, "(null)" will be printed. In all exponential formats, the default number of digits of exponent to display is three. Using the _set_output_format() function, the number of digits displayed may be set to two, expanding to three if demanded by the size of exponent. The %n format is inherently insecure and is disabled by default; if %n is encountered in a format string, the invalid parameter handler is invoked.