IBM C/C++ Tools S61G-1441-00

Reference Summary

Version 2.0

IBM C/C++ Tools S61G-1441-00

Reference Summary

Version 2.0

Second Edition (March 1993)

This edition applies to Version 2.0 of IBM C/C++ Tools (Programs 61G1176 and 61G1426) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition.

Requests for publications and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

Contents

This publication is a summary of information found in the *C Language Reference*, *C++ Language Reference*, *C Library Reference* and *Programming Guide*. It is intended as a quick reference for C and C++ programmers. For detailed information about the material found in this publication, please refer to the *Programming Guide* for compiler options, the *C Library Reference* for library functions, and to the language references for other constructs.

Character Set

a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
1 2 3 4 5 6 7 8 9

! " # % & ' ( ) * + , - . / :
; < = > ? [\ ] _ { | } ~ ^

The space character.

The control characters representing horizontal tab, vertical tab, form feed, and end of string.

Reserved Words

asm	enum	register	virtual
auto	extern	return	void
break	float	short	volatile
case	for	signed	while
catch	friend	sizeof	_Cdecl
char	goto	static	_Export
class	if	struct	_Far16
const	inline	switch	_Far32
continue	int	template	_Fastcall
default	long	this	_Inline
delete	new	throw	_Optlink
do	operator	try	_Packed
double	private	typedef	_Pascal
else	protected	union	_Seg16
	public	unsigned	_System

Escape Sequences

Escape Sequence	Character Represented
\a	Alert (bell)
\b	Backspace
\f	Form feed (new page)
\n	New-line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation
\"	Double quotation
\?	Question mark
\\	Backslash
\	Line continuation character (used to indicate the current line continues on the next line)

Storage Classes

auto	register	static	extern
_Inline			

Type Specifiers

char	double	signed	enum
int	long	unsigned	void
float	union	short	struct
long double			

Type Qualifiers

const	_Packed	_Seg16	volatile
_Export			

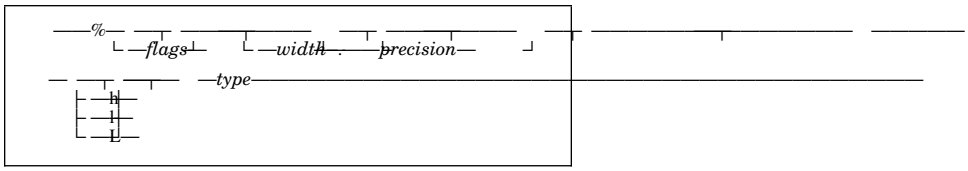
Linkage Specifiers

_Cdecl	_Far32	_Optlink	_System
_Far16	_Fastcall	_Pascal	

Operator Precedence

Precedence Level	Associativity	Operators
Primary	left to right	() [ ] . -> ::
Unary	right to left	++ -- - + ! ~ & (*typename*) sizeof . -> new delete
Multiplicative	left to right	/ %
Additive	left to right	+ -
Bitwise Shift	left to right	<< >>
Relational	left to right	< > <= >=
Equality	left to right	== !=
Bitwise Logical AND	left to right	&
Bitwise Exclusive OR	left to right	^ or ¬
Bitwise Inclusive OR	left to right	|
Logical AND	left to right	&&
Logical OR	left to right	||
Conditional	right to left	? :
Assignment	right to left	= += -= = /= <<= >>= %= &= ^= |=
Comma	left to right	,

printf Format Specifications



Field	Description
flags	Justification of output and printing of signs, blanks, decimal points, octal, and hexadecimal prefixes, and the semantics for wchar_t precision unit.
width	Minimum number of characters (bytes) output. You can specify width with an asterisk (*). If you specify an asterisk, the width comes from the next argument in the argument list.
precision	Maximum number of characters (bytes) printed for all or part of the output field, or minimum number of digits printed for integer values. You can specify precision with an asterisk (*). If you specify an asterisk, precision comes from the next argument in the argument list.
h,l,L	Size of argument expected: **h** A prefix, with the integer types d, i, n, o, u, x, and X, that specifies that the argument is a short int. **l** A prefix, with the types d, i, n, o, u, x, and X, that specifies that the argument is a long int.

L A prefix, with the types e, E, f, g, or G, that specifies that the argument is a long double.

printf Type Characters:

Character	Output Format
d, i	Signed decimal integer.
u	Unsigned decimal integer.
o	Unsigned octal integer.
x	Unsigned hexadecimal integer, using "abcdef".
X	Unsigned hexadecimal integer, using "ABCDEF".
e	Floating-point signed value having the form [–]*d.dddd*e[*sign*]*ddd*, where *d* is a single decimal digit, *dddd* is one or more decimal digits, *ddd* is at least two decimal digits, and *sign* is + or –.
E	Floating-point identical to the e format except that E introduces the exponent instead of e.
f	Floating-point signed value having the form [–]*dddd.dddd*, where *dddd* is one or more decimal digits.
F	Floating-point identical to the f format. Used in extended mode.
g	Floating-point 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 *precision*.
G	Floating-point identical to the g format except that E introduces the exponent (where appropriate) instead of e.
c	Single character.

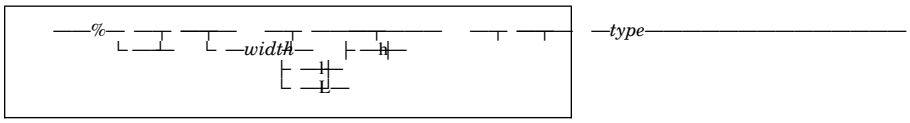
Character	Output Format
s	String of characters printed up to but not including the first null character (\ ) or until *precision* is reached.
n	Pointer to integer. Number of characters successfully written so far to the stream or buffer; this value is stored in the integer whose address is given as the argument. No value will be output.
p	Pointer to void converted to a sequence of printable characters.
lc	Multibyte character.
ls	Multibyte characters printed up to the first wchar_t null character (L'\ ') or until precision is reached.
%	A % is written. No argument is converted. The complete conversion specification is %%.

printf Flag Characters:

Flag	Meaning
–	Left-justifies the result within the field width.
+	Prefixes the output value with a sign (+ or –) if the output value is of a signed type.
blank(' ')	If the result of a signed conversion is a positive value or no characters, a space is prefixed to the result. If the + and the *blank* flags both appear, the *blank* flag is ignored. Positive signed value will be output with a sign.
0	Leading zeros are used to pad to the specified field width. If the 0 and - flags both appear, the 0 flag will be ignored. For d, i, o, u, x, and X conversions, if a precision is specified, the 0 flag will be ignored.

Flag	Meaning
#	When used with the o, x, or X formats, the # flag prefixes any nonzero output value with , x, or X, respectively. When used with the f, F, e, or E formats, the # flag forces the output value to contain a decimal point in all cases. When used with the g or G formats, the # flag forces the output value to contain a decimal point in all cases and prevents the truncation of trailing zeros. When used with the ls format, the # flag causes precision to be measured in wchar_t characters.

scanf Format Specifications



Field	Description
*	An asterisk following the percent sign suppresses assignment of the next input field, which is interpreted as a field of the specified *type*. The field is scanned but not stored.
width	Maximum number of characters to be read from stdin.
h	A prefix, with the integer types d, i, and n, that specifies that the argument is a short int. A prefix, with the integer types o, a, x, and X, that specifies that the argument is an unsigned short int. A prefix, with the types e, E, f, g, and G, that specifies that the argument is a float. The h modifier is ignored if specified for any other *type*.
l	A prefix, with the integer types d, i, and n, that specifies that the argument is a long int. A prefix, with the integer types o, u, x, and X, that specifies that the argument is an unsigned long int. A prefix, with the types e, E, f, g, and G, that specifies that the argument is a double. The l modifier is ignored if specified for any other *type*.
L	A prefix, with the types e, E, f, g, and G, that specifies that the argument is a long double.

scanf Type Characters

Character	Type of Input Expected
d	Decimal integer.
o	Octal integer.
x,X	Hexadecimal integer.
i	Decimal, hexadecimal, or octal integer.
u	Unsigned decimal integer.
e f g	Floating-point value consisting of an optional sign (+ or –), a series of one or more decimal digits possibly containing a decimal point, and an optional exponent (e or E) followed by an integer value that may be signed.
E, F, G	Floating-point identical to the e, f, and g formats, except they are only used in extended mode.
c	Character; white-space characters that are ordinarily skipped are read when c is specified.
s	String of characters scanned up to the first white-space character or until *precision* is reached.
n	No input read from *stream* or buffer. Pointer to integer, where the number of characters successfully read from the stream or buffer up to that point in the call to scanf are stored.
lc	Multibyte character constant.
ls	Multibyte string constant.
p	Pointer to void converted to series of characters.
[]	Strings of characters that are part of the scan set are scanned.
%	% matches a single %; no conversion or assignment occurs. The complete conversion specification is %%.

C++ Input and Output

For details on C++ I/O, see the *Standard Class Library Reference*.

Operators:

>>	Input or read
<<	Output or write

Formatting Variables and Flags: The following variables and flags are all declared in the ios class.

Formatting Variables	
short x_precision;	The number of significant digits in the representation of floating-point values. The default value is 6.
char x_fill;	The character used to pad values. The default value is a space character.
short x_width;	The minimum width of a field. The default value is 0.
White Space and Padding	
skipws	White space is skipped on input.
left	The value is left-justified and fill characters are added after the value.
right	The value is right-justified and fill characters are added before the value.
internal	Fill characters are added after any leading sign or base notation but before the value itself.
Base Conversion	
dec	The conversion base is 10.
oct	The conversion base is 8.
hex	The conversion base is 16.

showbase	Values are converted to an external form that can be read according to the C++ lexical conventions for integral constants.
Integral Formatting	
showpos	A positive sign (+) is inserted into decimal conversions of positive integral values.
Floating-Point Formatting	
fixed	Values are converted to fixed notation.
scientific	Values are converted using scientific notation.
showpoint	Trailing zeros and a decimal point appear in the result of a floating-point conversion.
Case	
uppercase	An uppercase E is used for floating-point values in scientific notation, hexadecimal digits A to H are stored in uppercase, and an uppercase X is placed before hexadecimal values when showbase is set. If uppercase is not set, lowercase letters are used.
Buffer Flushing	
stdio	Used internally by the sync_with_stdio function to allow I/O functions from the I/O Stream library and <**stdio.h**> to be used in the same program.
unitbuf	The buffer is flushed after each insertion, and the attached stream buffer is unit-buffered.

| The formatting flags and variables can be set and unset using member functions of the ios class or using the parameterized manipulators. For example, to set x_precision to 4 for an output statement for a floating-point number fp, you could use either:

```
cout.precision(4);
cout << fp;
```

| or

```
cout << setprecision(4) << fp;
```

| You can also call the member functions outside of the actual I/O statements.

| The built-in ios manipulators can also be used to set formatting flags. For example, to set dec on input, you could use the following statement:

```
cin >> dec;
```

| The member functions and manipulators are described in the following section.

| **Member Functions for Formatting:** The version of the function that takes no arguments returns the value of the current format state.

| char fill();
| char fill(char *fillchar*);
| Sets the fill character (x_fill). The position of the fill character is determined by the left, right, and internal flags.
| long flags() const;
| long flags(long *flagset*);
| Sets the flags to the setting specified in *flagset*.
| int precision() const;
| int precision(int *prec*);
| Sets the number of significant digits to be used for floating-point values (x_precision).

| long setf(long *newset*);
| long setf(long *newset*, long *field*);
| When only one parameter is specified, sets the flags to the setting specified in *newset*. When two parameters are specified, clears the flags specified in *field* and resets them to the settings in *newset*.
| int skip(int *i*);
| Sets the format flag skipws.
| long unsetf(long *oflags*);
| Turns off the format flags specified in *oflags* and returns the previous format state.
| int width() const;
| int width(int *fwidth*);
| Sets the minimum field width (x_width).

| **Manipulators:** The following built-in manipulators can modify the formatting flags of ios objects:

| dec Sets dec.
| hex Sets hex.
| oct Sets oct.
| ws Extracts white space characters from an input stream buffer. Valid for input only.
| endl Inserts a new-line character and calls flush(). Valid for output only.
| ends Inserts a null character. Valid for output only.
| flush Flushes the output stream buffer. Valid for output only.

| The following parameterized manipulators defined in <**iomanip.h**> can modify the format state of ios objects:

| resetiosflags(long *flags*);
| Clears the format flags specified in *flags*.
| setbase(int *base*);
| Sets the conversion base to *base*.
| setfill(int *fill*);
| Sets the fill character (x_fill) to *fill*).

| setiosflags(long *flags*);
| Sets the format flags specified in *flags*.

| setprecision(int *prec*);
| Sets the precision variable (x_precision) to *prec*.

| setw(int *width*);
| Sets the minimum field width (x_width) to *width*.

| **Checking the Error State:** The following error bits are used in the error state for ios objects:

| badbit
| eofbit
| failbit
| goodbit
| hardfail

| Do not use the hardfail bit; it is used internally by the I/O Stream Library.

| The error bits can be accessed using the following member functions of the ios class:

| int bad() const;
| Returns a nonzero value if badbit is set. The badbit bit is usually set when some operation on the streambuf object associated with the ios object has failed. I/O operations will probably not continue for the ios object.

| void clear(int *state* =); Changes the error state to *state*. If *state* equals (the default), all bits in the error state are cleared. You can use the bitwise OR operator (|) to set one bit without clearing the others. For example, to set only badbit in *iosobj* and retain the bits already set, use the following statement:
| *iosobj*.clear(ios::badbit|*iosobj*.rdstate());

| int eof() const; Returns a nonzero value if eofbit is set. The eofbit bit is usually set when the end-of-file has been encountered during a read operation.

| int fail() const; Returns a nonzero value if badbit or failbit is set.

| int good() const; Returns a nonzero value if any bits are set in the error state.

| int rdstate() const; Returns the current value of the error state.

| operator void (); operator const void () const; Converts an ios object to a pointer so it can be compared to . Returns if badbit or failbit is set.

| int operator!() const; Returns a nonzero value if badbit or failbit is set.

Compiler Options

Option	Description	Default
/B"*options*"	Specify parameters to pass to linker.	/B"" Pass no parameters to the linker.
/C[+|-]	Perform compile only, or compile and link.	/C- Perform compile and link.
/D*name*[<=|::>*n*]	Define preprocessor macros to specified values.	Define no macros on the command line.
/Fa[+|-] /Fa*name*	Produce and name an assembler listing file that has the source code as comments.	/Fa- Do not create an assembler listing file.
/Fb[+|-]	Produce a browser listing file.	/Fb- Do not produce a browser listing file.
/Fc[+|-]	Perform syntax check only.	/Fc- Compile and produce output files according to other options.
/Fd[+|-]	Specify work file storage area.	/Fd- Store internal work files in shared memory.
/Fe*name*	Specify name of executable file or DLL.	Give the file the same name as the source file, with the extension .EXE or .DLL.

Option	Description	Default
/Fi[+|-]	Control creation of precompiled header files.	/Fi- Do not create a precompiled header file.
/Fl[+|-] /Fl*name*	Specify name of listing file.	/Fl- Give the listing the same file name as the source file, with the extension .LST.
/Fm[+|-] /Fm*name*	Produce a linker map file. When used with the /C option, the /Fm option has no effect.	/Fm- Do not create map file.
/Fo[+|-] /Fo*name*	Suppress creation or specify name of object file.	/Fo+ Create object file with the same name as the source file and the extension .OBJ.
/Ft[+|-] /Ft*dir*	Control generation of files for template resolution. **Valid for C++ files only.**	/Ft+ Generate files for template resolution in the TEMPINC directory.
/Fw[+|-] /Fw*name*	Control generation and use of intermediate files.	/Fw- Perform regular compilation; do not save intermediate files.
/Gd[+|-]	Specify dynamic or static linking of runtime libraries.	/Gd- Statically link the runtime library.
/Ge[+|-]	Specify creation of .EXE or .DLL files.	/Ge+ Create an .EXE file.

Option	Description	Default
/Gf[+|-]	Generate code for fast floating-point execution.	/Gf- Do not generate code for fast floating-point execution.
/Gh[+|-]	Generate code enabled for EXTRA and other profiling tools.	/Gh- Do not enable code for EXTRA.
/Gi[+|-]	Generate code for fast integer execution.	/Gi- Do not generate code for fast integer execution.
/Gm[+|-]	Choose single or multithread libraries.	/Gm- Link with the single-thread version of the library (no multithread support).
/Gn[+|-]	Control the generation of default libraries in object files.	/Gn- Generate default library search, according to other /G options specified.
/Gr[+|-]	Control generation of code that can run at ring 0.	/Gr- Do not generate code that can run at ring 0.
/Gs[+|-]	Remove stack probes from generated code.	/Gs- Do not remove stack probes.
/Gt[+|-]	Align 16-bit objects within 64K boundaries.	/Gt- Normal alignment of objects.

Option	Description	Default
/Gu[+|-]	Tell intermediate linker whether data defined in the intermediate link will be used by external functions not defined in the intermediate link.	/Gu- Tell intermediate linker that data in the intermediate files being linked could be used by external functions.
/Gv[+|-]	Control whether DS and ES registers are saved and restored on external function calls for virtual device driver development. **Valid for C files only.**	/Gv- Do not perform any special handling of DS and ES registers.
/Gw[+|-]	Control generation of FWAIT instruction after each floating-point load instruction.	/Gw- Do not generate FWAIT instruction after each floating-point load instruction.
/Gx[+|-]	Remove C++ exception handling information. **Valid for C++ files only.**	/Gx- Do not remove C++ exception handling information.
/G<3|4|5>	Specify processor to optimize for.	/G3 Optimize code for the 386 processor.
/H*num*	Set significant length of external names.	/H255 Set the first 255 characters to be significant.

Option	Description	Default
/I*path[;path]*	Specify #include search path(s).	/I- Search directory of source file, and then paths given in the INCLUDE environment variable.
/J[+|-]	Set default char type.	/J+ Set unspecified char variables to unsigned char.
/Ka[+|-]	Control messages about long variable assignments causing precision loss. **Valid for C files only. Maps to the** /Wtrd **option.**	/Ka- Suppress the messages.
/Kb[+|-]	Control basic diagnostic messages generated by /K options. **Valid for C files only. Maps to the** /Wgen **option.**	/Kb- Suppress basic diagnostic messages.
/Kc[+|-]	Control preprocessor warning messages. **Valid for C files only. Maps to the** /Wppc **option.**	/Kc- Suppress the messages.
/Ke[+|-]	Control messages about enum usage. **Valid for C files only. Maps to the** /Wenu **option.**	/Ke- Suppress the messages.

Option	Description	Default
/Kf[+|-]	Set all diagnostic messages on or off. **Valid for C files only. Maps to the** /Wall **option.**	/Kf- Suppress all the messages.
/Kg[+|-]	Control messages about the appearance and usage of goto statements. **Valid for C files only. Maps to the** /Wgot **option.**	/Kg- Suppress the messages.
/Ki[+|-]	Control messages about uninitialized variables. **Valid for C files only. Maps to the** /Wini **and** /Wuni **options.**	/Ki- Suppress the messages.
/Ko[+|-]	Control portability messages. **Valid for C files only. Maps to the** /Wpor **option.**	/Ko- Suppress the messages.
/Kp[+|-]	Control messages about unused function parameters. **Valid for C files only. Maps to the** /Wpar **option.**	/Kp- Suppress the messages.
/Kr[+|-]	Control messages about name mapping. **Valid for C files only. Maps to the** /Wtru **option.**	/Kr- Suppress the messages.

Option	Description	Default
/Kt[+|-]	Control preprocessor trace messages. **Valid for C files only.** Maps to the /Wppt option.	/Kt- Suppress the messages.
/Kx[+|-]	Control messages about unreferenced external variables. **Valid for C files only.** Maps to the /Wext and /Wuse options.	/Kx- Suppress the messages.
/L[+|-]	Produce a listing file.	/L- Do not produce a listing file.
/La[+|-]	Include layout of all struct and union variables referenced by the user, with offsets and lengths.	/La- Do not include layout.
/Lb[+|-]	Include layout of all struct and union variables declared in the source.	/Lb- Do not include layout.
/Le[+|-]	Expand all macros.	/Le- Do not expand macros.
/Lf[+|-]	Set all listing options on or off.	/Lf- Set all listing options off.
/Li[+|-]	Expand user #include files.	/Li- Do not expand user #include files.

Option	Description	Default
/Lj[+|-]	Expand user and system #include files.	/Lj- Do not expand user and system #include files.
/Lp*num*	Set page length.	/Lp66 Set page length to 66 lines.
/Ls[+|-]	Include the source code.	/Ls- Do not include the source code.
/Lt"*string*"	Set title string.	Set the title to be the name of the source file.
/Lu"*string*"	Set subtitle string.	/Lu"" Set no subtitle (null string).
/Lx[+|-]	Generate a cross-reference table of names that are referenced.	/Lx- Do not generate a cross-reference table.
/Ly[+|-]	Generate a complete cross-reference table of all variable, structure, and function names.	/Ly- Do not generate a cross-reference table.
/Mp|/Ms	Set _System or _Optlink calling convention.	/Mp Use the _Optlink calling convention.
/N*n*	Set maximum number of errors before compilation ends.	Set no limit on number of errors.
/Nd*name*	Specify names of default data and constant segments.	Use the default names DATA32 and CONST32.

Option	Description	Default
/Nt*name*	Specify name of default text segment.	Use the default name CODE32.
/O[+|-]	Turn on optimization.	/O- Do not optimize code.
/Oi[+|-|] /Oi*value*	Control inlining of user functions.	/Oi- Do not inline user functions. **Note:** If /O+ is specified, /Oi+ (inline user functions qualified by _Inline or inline) becomes the default.
/Ol[+|-]	Control use of the intermediate code linker.	/Ol- Do not use the intermediate code linker during code generation.
/Om[+|-]	Control size of the working set.	/Om- Do not limit the working-set size.
/Op[+|-]	Control optimizations involving the instruction pointer.	/Op+ Perform optimizations involving the instruction pointer.
/Os[+|-]	Control use of the instruction scheduler.	/Os- Do not invoke the instruction scheduler. **Note:** If /O+ is specified, /Os+ (use the instruction scheduler) becomes the default. You cannot specify /Os+ with /O-.

Option	Description	Default
/P[+|-]	Control the preprocessor.	/P- Compile and link as specified by other options.
/Pc[+|-]	Preserve comments in preprocessor output.	/Pc- Remove comments from preprocessor output.
/Pd[+|-]	Redirect preprocessor output .	/Pd- Write preprocessor output to a file with the same name as the source file and the extension .I.
/Pe[+|-]	Suppress generation of #line directives in preprocessor output.	/Pe- Generate #line directives in preprocessor output.
/Q[+|-]	Control compiler logo display.	/Q- Display logo on stderr.
/Re|/Rn	Control runtime environment.	/Re Generate executable code to run in a C/C++ Tools runtime environment.
/S[a|c|e|2]	Set language level. /Sa= only ANSI constructs. /Sc= constructs allowed in earlier versions of C++. /S2= only SAA* Level 2 constructs.	/Se Allow all language constructs.
/Sd[+|-]	Set default file extension.	/Sd- Set the default file extension as .OBJ.

Option	Description	Default
/Sg[*l*][,<*r*|>] /Sg-	Set left and right margins of the input file. **Valid for C files only.**	/Sg- Use no margins.
/Sh[+|-]	Enable ddname support.	/Sh- Do not allow ddnames.
/Si[+|-]	Control use of precompiled header files.	/Si+ Use precompiled header files if they exist and are current.
/Sm[+|-]	Control compiler interpretation of unsupported keywords.	/Sm- Treat unsupported keywords like any other identifier.
/Sn[+|-]	Allow DBCS use.	/Sn- Do not allow DBCS.
/Sp[1|2|4]	Specify alignment of data within structures and unions.	/Sp4 Align data on natural boundaries; strictest alignment is a 4-byte boundary.
/Sq[*l*][,<*r*|>] /Sq-	Specify columns in which sequence numbers appear. **Valid for C files only.**	/Sq- Use no sequence numbers.
/Sr[+|-]	Set type conversion rules to preserve accuracy or sign.	/Sr- Preserve accuracy.
/Ss[+|-]	Allow use of double slashes (//) for comments. **Valid for C files only.**	/Ss- Do not allow double slashes to indicate comments.

Option	Description	Default
/Su[+|-|1|2|4]	Control size of enum variables.	/Su- Make the type of enum variables the smallest integral type that can contain all enum values.
/Sv[+|-]	Allow use of memory files.	/Sv- Do not allow memory files.
/Tc	Compile the following file as a C source file. This option must be immediately followed by a file name.	Compile .cpp and .cxx files as C++ files, and all other unrecognized files (not .obj, .def, .lib, or one of the .w files) as C files.
/Td[c|p]	Specify default language (C or C++) for source files.	/Td Compile .cpp and .cxx files as C++ files, and all other unrecognized files as C files.
/Ti[+|-]	Generate debugger information.	/Ti- Do not generate debugger information.
/Tp	Compile the following file as a C++ source file. This option must be immediately followed by a file name.	Compile .cpp and .cxx files as C++ files, and all other unrecognized files (not .obj, .def, .lib, or one of the .w files) as C files.

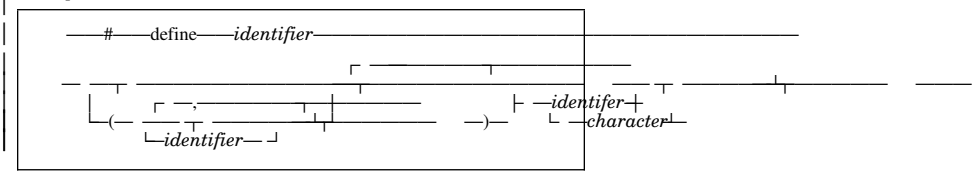
Option	Description	Default
/Ts[+|-]	Generate code to allow the debugger to maintain the call stack across all calls that do not chain the EBP.	/Ts- Do not generate code to allow the debugger to maintain the call stack.
/Tx[+|-]	Control information generated when an exception occurs.	/Tx- Provide only the exception message and the address where the exception occurs; do not provide a machine-state dump.
/U<*name*| >	Remove macros.	Retain macros.
/V"*string*"	Include a version string in object file.	/V"" Set no version string.
/W*grp*[+|-]	Control diagnostic messages. See Diagnostic (/W*grp*) Options in the next section for the explanation of the *grp* groupings.	Do not generate diagnostic messages.
/W[|1|2|3]	Set the type of message the compiler produces.	/W3 Produce all message types.
/Xc[+|-]	Ignore search paths specified using /I.	/Xc- Search paths specified using /I.
/Xi[+|-]	Control INCLUDE environment variable search paths.	/Xi- Search paths specified in the INCLUDE environment variable.

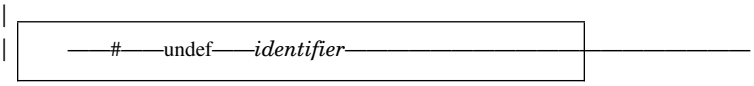
Diagnostic (/W*grp*) Options

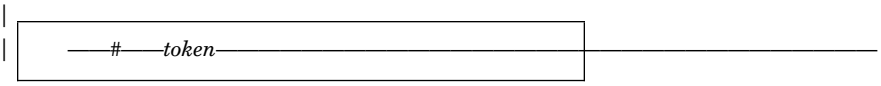
grp Value	Controls Messages About
all	All diagnostics.
cls	Use of classes.
cmp	Possible redundancies in unsigned comparisons.
cnd	Possible redundancies or problems in conditional expressions.
cns	Operations involving constants.
cnv	Conversions.
cpy	Copy constructors that cannot be generated.
dcl	Consistency of declarations.
eff	Statements with no effect.
enu	Consistency of enum variables.
ext	Unused external definitions.
gen	General diagnostics.
gnr	Generation of temporary variables.
got	Usage of goto statements.
ini	Possible problems with initialization.
lan	Effects of the language level.
obs	Features that are obsolete.
ord	Unspecified order of evaluation.
par	Unused parameters.
por	Language constructs that are not portable.
ppc	Possible problems with using the preprocessor.
ppt	Trace of preprocessor actions.
pro	Missing function prototypes.
rea	Code that cannot be reached.
ret	Consistency of return statements.

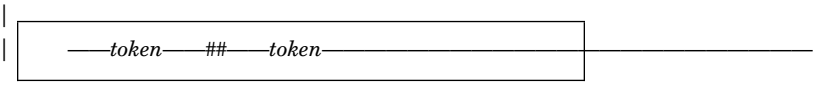
grp **Value**	**Controls Messages About**
trd	Possible truncation or loss of data.
tru	Variable names truncated by the compiler.
und	Casting of pointers to or from an undefined class.
uni	Uninitialized variables.
use	Unused auto and static variables.
vft	Generation of virtual function tables.

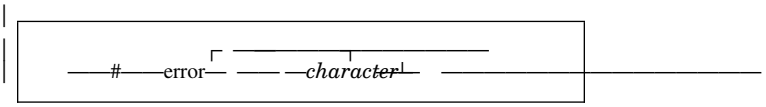
Preprocessor Directives

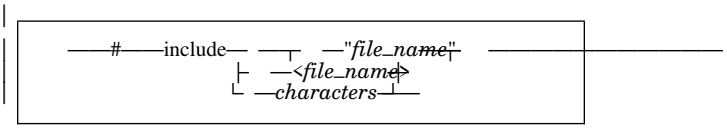


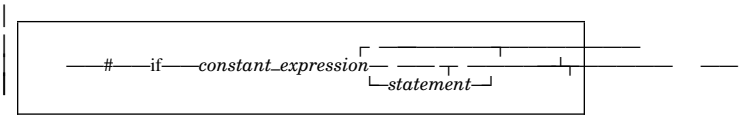


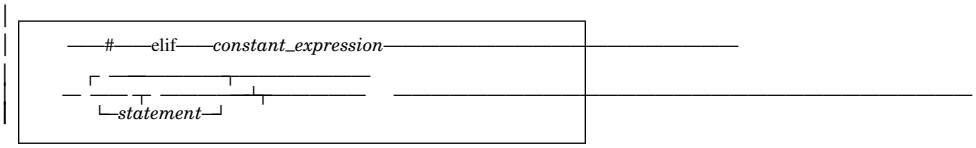


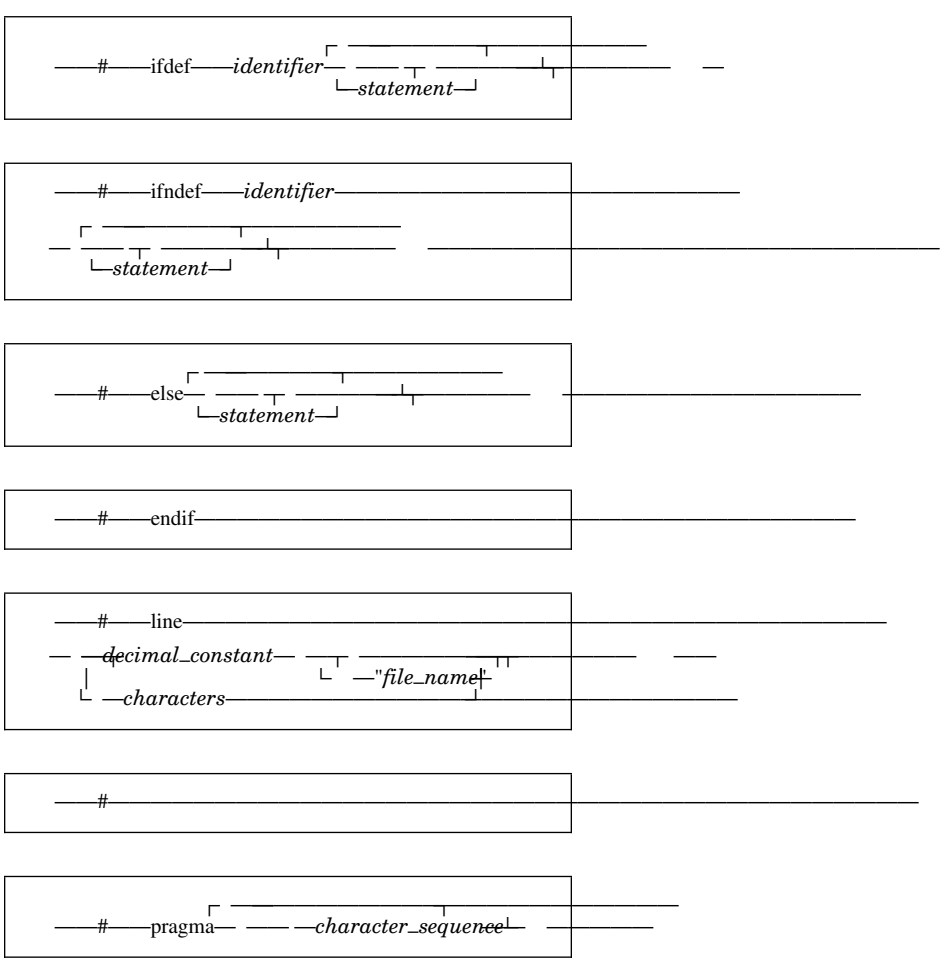






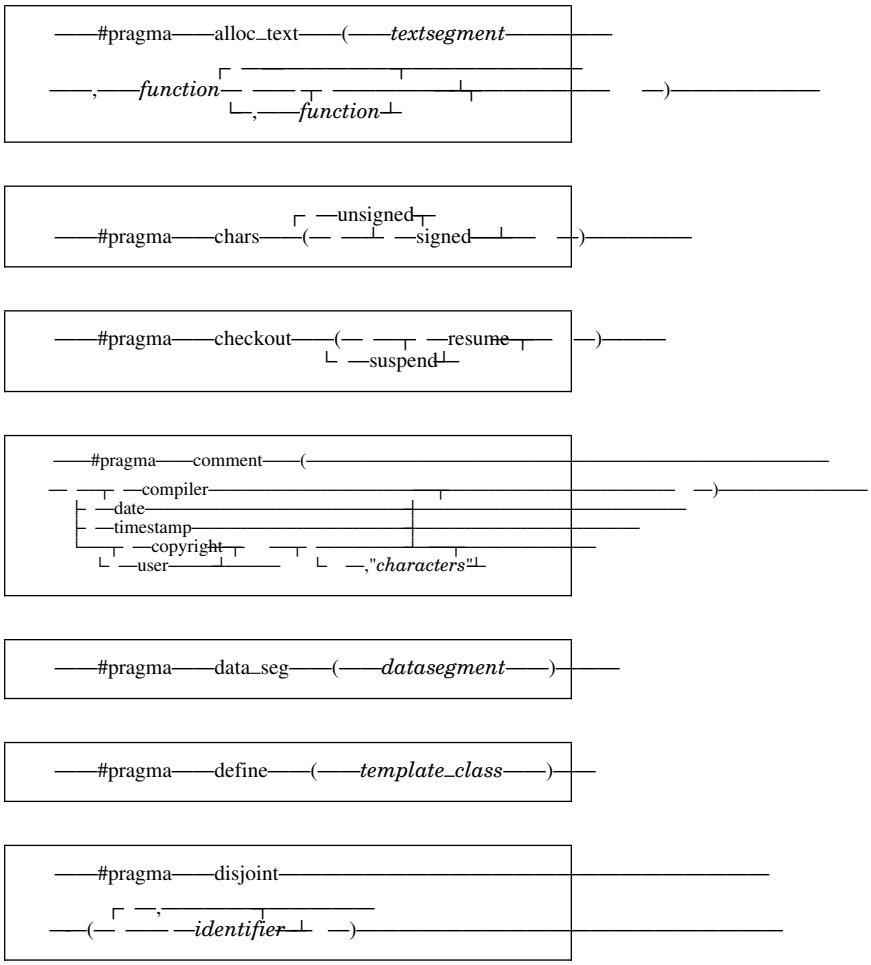




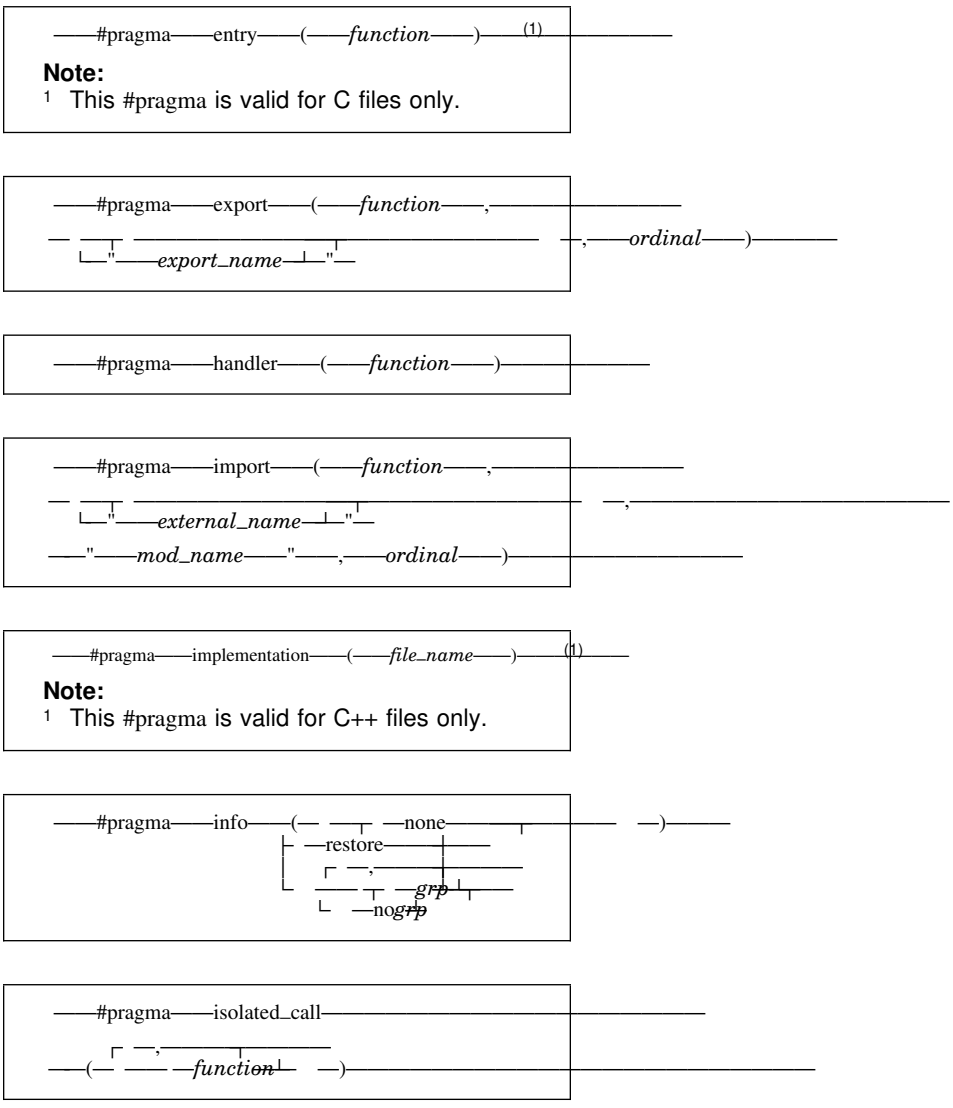


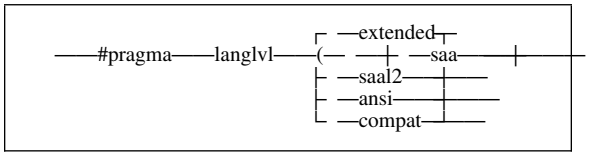
The #pragma directives are listed in the following section.

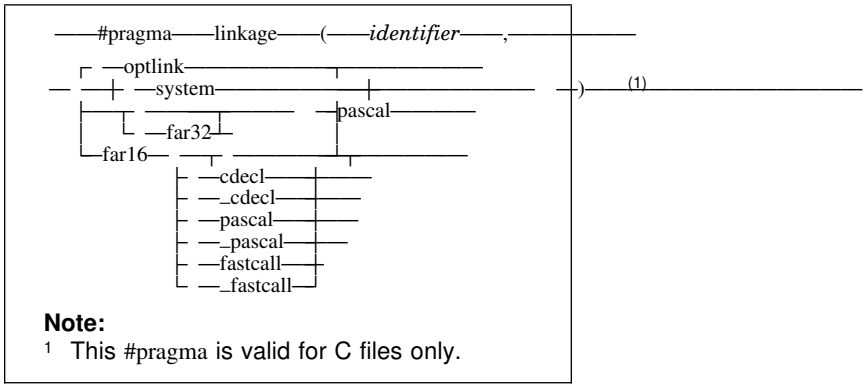
Pragma Directives



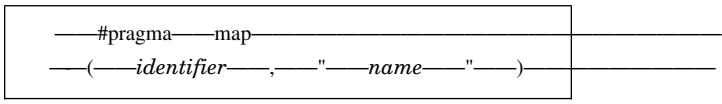
/

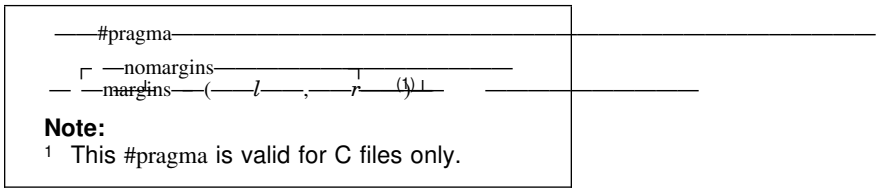

#pragma entry (function) (1)
Note:
1 This #pragma is valid for C files only.
#pragma export (function , " export_name " , ordinal)
#pragma handler (function)
#pragma import (function , " external_name " , " mod_name " , ordinal)
#pragma implementation (file_name) (1)
Note:
1 This #pragma is valid for C++ files only.
#pragma info (none restore , grp nogrp)
#pragma isolated_call (, function)



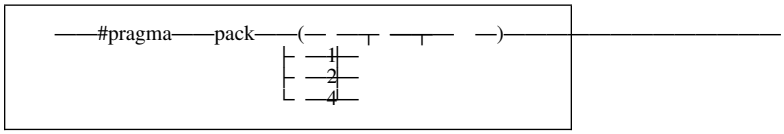


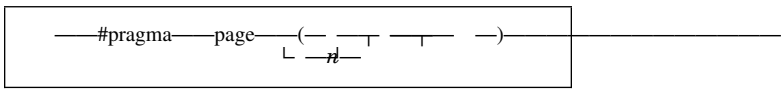
Note:
[1] This #pragma is valid for C files only.

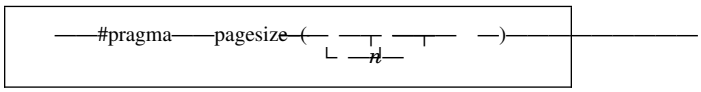


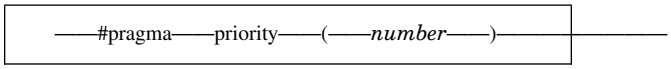


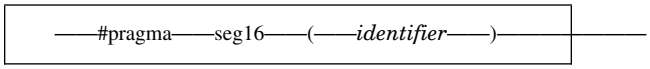
Note:
[1] This #pragma is valid for C files only.

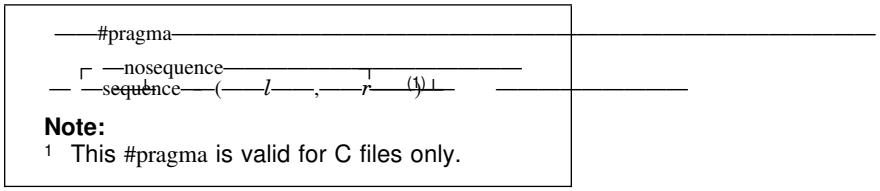




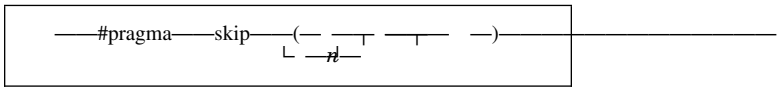


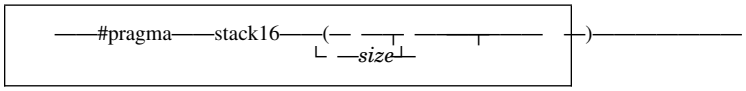


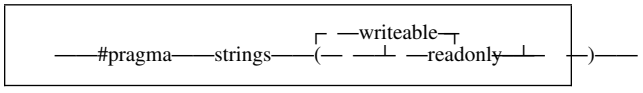


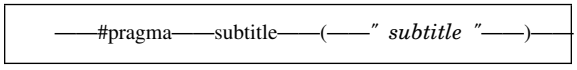


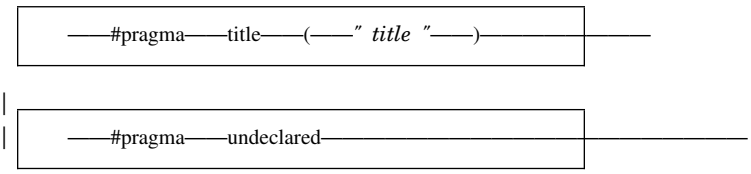
Note:
[1] This #pragma is valid for C files only.










#pragma
title
(
" title "
)
#pragma
undeclared

Library Functions

Function and Parameters	SAA Function?	Header
void abort(void);	√	<stdlib.h> <process.h>
int abs(int *n*);	√	<stdlib.h>
int _access(char *pathname*, int *mode*);		<io.h>
double acos(double *x*);	√	<math.h>
void _alloca(size_t *size*);		<stdlib.h> <malloc.h> <builtin.h>
char asctime(const struct tm *time*);	√	<time.h>
double asin(double *x*);	√	<math.h>
void assert(int *expression*);	√	<assert.h>
double atan(double *x*);	√	<math.h>
double atan2(double *y*, double *x*);	√	<math.h>
int atexit(void (*func*)(void));	√	<stdlib.h>
double atof(const char *string*);	√	<stdlib.h>
int atoi(const char *string*);	√	<stdlib.h>
long int atol(const char *string*);	√	<stdlib.h>
long double _atold(const char *nptr*);		<stdlib.h> <math.h>
int _beginthread(void (*start_address*)(void), (void)*stack*, unsigned *stack_size*, void *argument_list*);		<stdlib.h> <process.h>

Function and Parameters	SAA Function?	Header
void bsearch(const void *key*, const void *base*, size_t *num*, size_t *size*, int (*comp*) (const void *element1*, const void *element2*));	√	<stdlib.h> <search.h>
double _cabs(struct complex *z*);		<math.h>
void calloc(size_t *num*, size_t *size*);	√	<stdlib.h> <malloc.h>
double ceil(double *x*);	√	<math.h>
char _cgets(char *str*);		<conio.h>
int _chdir(char *pathname*);		<direct.h>
int _chdrive(int *drive*);		<direct.h>
int _chmod(char *pathname*, int *mode*);		<io.h> <sys\stat.h>
int _chsize(int *handle*, long int *size*);		<io.h>
unsigned int _clear87(void);		<float.h>
void clearerr(FILE *stream*);	√	<stdio.h>
clock_t clock(void);	√	<time.h>
int _close(int *handle*);		<io.h>
unsigned int _control87(unsigned int *new,* unsigned int *mask*);		<float.h>
double cos(double *x*);	√	<math.h>
double cosh(double *x*);	√	<math.h>
int _cprintf(char *format-string*, *argument_list*);		<conio.h>
int _cputs(char *str*);		<conio.h>

Function and Parameters	SAA Function?	Header
int _creat(char *pathname*, int *mode*);		<io.h> <sys\stat.h>
int _cscanf(char *format-string*, *argument_list*);		<conio.h>
char ctime(const time_t *time*);	√	<time.h>
int _cwait(int *stat_loc*, int *process_id*, int *action_code*);		<process.h>
void _debug_calloc(size_t *num*, size_t *size*, const char *pFilename*, size_t *linenum*);		<stdlib.h> <malloc.h>
void _debug_free(void *ptr*, const char *pFilename*, size_t *linenum*);		<stdlib.h> <malloc.h>
int _debug_heapmin(const char *pFilename*, size_t *linenum*);		<stdlib.h> <malloc.h>
void _debug_malloc(size_t *size*, const char *pFilename*, size_t *linenum*);		<stdlib.h> <malloc.h>
void _debug_realloc(void *ptr*, size_t *size*, const char *pFilename*, size_t *linenum*);		<stdlib.h> <malloc.h>
double difftime(time_t *time2*,time_t *time1*);	√	<time.h>
void _disable(void);		<stdlib.h> <builtin.h>

Function and Parameters	SAA Function?	Header
div_t div(int *num*, int *denom*);	√	<stdlib.h>
void _dump_allocated(int *nBytes*);		<stdlib.h> <malloc.h>
int _dup(int *handle*);		<io.h>
int _dup2(int *handle1*, int *handle2*);		<io.h>
char _ecvt(double *value*, int *ndigits*, int *decptr*, int *signptr*);		<stdlib.h>
void _enable(void);		<stdlib.h> <builtin.h>
void _endthread(void);		<stdlib.h> <process.h>
int _eof(int *handle*);		<io.h>
double erf(double *x*);	√	<math.h>
double erfc(double *x*);	√	<math.h>
int _execl(char *pathname*, char *arg* , char *arg1*,..., char *argn*, NULL);		<process.h>
int _execle(char *pathname*, char *arg* , char *arg1*,..., char *argn*, NULL char *envp*[]);		<process.h>
int _execlp(char *pathname*, char *arg* , char *arg1*,..., char *argn*, NULL);		<process.h>
int _execlpe(char *pathname*, char *arg* , char *arg1*,..., char *argn*, NULL, char *envp*[]);		<process.h>

Function and Parameters	SAA Function?	Header
int _execv(char *pathname*, char *argv*[]);		<process.h>
int _execve(char *pathname*, char *argv*[], char *envp*[ ]);		<process.h>
int _execvp(char *pathname*, char *argv*[]);		<process.h>
int _execvpe(char *pathname*, char *argv*[], char *envp*[]);		<process.h>
void exit(int *status*);	√	<stdlib.h> <process.h>
void _exit(int *status*);		<stdlib.h> <process.h>
double exp(double *x*);	√	<math.h>
double fabs(double *x*);	√	<math.h>
double _facos(double *x*);		<builtin.h> <math.h>
double _fasin(double *x*);		<builtin.h> <math.h>
int fclose(FILE *stream*);	√	<stdio.h>
int _fcloseall(void);		<stdio.h>
double _fcos(double *x*);		<builtin.h> <math.h>
double _fcossin(double *x*, double *y*);		<builtin.h>
char _fcvt(double *value*, int *ndec*, int *decptr*, int *signptr*);		<stdlib.h>
FILE _fdopen(int *handle*, char *type*);		<stdio.h>

Function and Parameters	SAA Function?	Header
int feof(FILE *stream*);	√	<stdio.h>
int ferror(FILE *stream*);	√	<stdio.h>
int fflush(FILE *stream*);	√	<stdio.h>
int fgetc(FILE *stream*);	√	<stdio.h>
int _fgetchar(void);		<stdio.h>
int fgetpos(FILE *stream*, fpos_t *pos*);	√	<stdio.h>
char fgets (char *string*,int *n*, FILE *stream*);	√	<stdio.h>
long int _filelength(int *handle*);		<io.h>
int _fileno(FILE *stream*);		<stdio.h>
double floor(double *x*);	√	<math.h>
int _flushall(void);		<stdio.h>
double fmod(double *x*, double *y*);	√	<math.h>
FILE fopen(const char *filename*, const char *type*);	√	<stdio.h>
double _fpatan(double *x*);		<builtin.h> <math.h>
void _fpreset(void);		<float.h>
int fprintf(FILE *stream*, const char *format*, *argument_list*);	√	<stdio.h>
double _fptan(double *x*);		<builtin.h> <math.h>
int fputc(int *c*, FILE *stream*);	√	<stdio.h>

Function and Parameters	SAA Function?	Header
int _fputchar(int *c*);		<stdio.h>
int fputs(const char *string*, FILE *stream*);	√	<stdio.h>
size_t fread(void *buffer*, size_t *size*, size_t *count*, FILE *stream*);	√	<stdio.h>
void free(void *ptr*)	√	<stdlib.h> <malloc.h>
int _freemod(unsigned long *module_handle*);		<stdlib.h>
FILE freopen(const char *filename*, const char *type*,FILE *stream*);	√	<stdio.h>
double frexp(double *x*, int *expptr*);	√	<math.h>
int fscanf(FILE *stream*, const char *format*, *argument_list*);	√	<stdio.h>
int fseek(FILE *stream*, long int *offset*, int *origin*);	√	<stdio.h>
int fsetpos(FILE *stream*, const fpos_t *pos*);	√	<stdio.h>
double _fsin(double *x*);		<builtin.h> <math.h>
double _fsincos(double *x*, double *y*);		<builtin.h>
double _fsqrt(double *x*)		<builtin.h>
int _fstat(int *handle*, struct stat *buffer*);		<stat.h> <sys\stat.h>
long int ftell(FILE *stream*);	√	<stdio.h>

Function and Parameters	SAA Function?	Header
void _ftime(struct timeb *timeptr*);		<timeb.h> <sys\timeb.h>
char _fullpath(char *pathbuf*, char *partialpath,* size_t *n*);		<stdlib.h>
size_t fwrite(const void *buffer*, size_t *size*, size_t *count*, FILE **stream*);	√	<stdio.h>
double _fyl2x(double *x*, double *y*);		<builtin.h>
double _fyl2xp1(double *x*, double *y*);		<builtin.h>
double _f2xm1(double *x*);		<builtin.h>
double gamma(double *x*);	√	<math.h>
char _gcvt(double *value*, int *ndec*, char *buffer*);		<stdlib.h>
int getc(FILE *stream*);	√	<stdio.h>
int _getch(void);		<conio.h>
int getchar(void);	√	<stdio.h>
int _getche(void);		<conio.h>
char _getcwd(char *pathbuf*, int *n*);		<direct.h>
char _getdcwd(int *drive*, char *pathbuf*, int *n*);		<direct.h>
int _getdrive(void);		<direct.h>
char getenv(const char *varname*);	√	<stdlib.h>
int _getpid(void);		<process.h>
unsigned long _getTIBvalue(const unsigned int *offset*);		<builtin.h>

Function and Parameters	SAA Function?	Header
char gets(char *buffer*);	√	<stdio.h>
struct tm gmtime(const time_t *time*);	√	<time.h>
void _heap_check(void);		<stdlib.h> <malloc.h>
int _heapmin(void);		<stdlib.h> <malloc.h>
double hypot(double *side1*, double *side2*);	√	<math.h>
int _inp(const unsigned int *port*);		<conio.h> <builtin.h>
unsigned long _inpd(const unsigned int *port*);		<conio.h> <builtin.h>
unsigned short _inpw(const unsigned int *port*);		<conio.h> <builtin.h>
void _interrupt(const unsigned int *num*);		<builtin.h>
int isalnum(int *c*);	√	<ctype.h>
int isalpha(int *c*);	√	<ctype.h>
int _isascii(int *c*);		<ctype.h>
int _isatty(int *handle*);		<io.h>
int iscntrl(int *c*);	√	<ctype.h>
int _iscsym(int *c*);		<ctype.h>
int _iscsymf(int *c*);		<ctype.h>
int isdigit(int *c*);	√	<ctype.h>
int isgraph(int *c*);	√	<ctype.h>
int islower(int *c*);	√	<ctype.h>
int isprint(int *c*);	√	<ctype.h>
int ispunct(int *c*);	√	<ctype.h>

Function and Parameters	SAA Function?	Header
int isspace(int *c*);	√	<ctype.h>
int isupper(int *c*);	√	<ctype.h>
int isxdigit(int *c*);	√	<ctype.h>
char _itoa(int *value*, char *string*, int *radix*);		<stdlib.h>
int _kbhit(void);		<conio.h>
double _j (double *x*);	√	<math.h>
double _j1(double *x*);	√	<math.h>
double _jn(int *n*, double *x*);	√	<math.h>
long int labs(long int *n*);	√	<stdlib.h>
double ldexp(double *x*, int *exp*);	√	<math.h>
ldiv_t ldiv(long int *numerator*, long int *denominator*);	√	<stdlib.h>
char _lfind(char *search_key*, char *base*, unsigned int *num*, unsigned int *width*, int ( )(const void *key*, const void *element*));		<search.h>
char _lsearch(char *search_key*, char *base*, unsigned int *num*, unsigned int *width*, int ( )(const void *key*, const void *element*));		<search.h>
int _loadmod(char *module_name*, unsigned long *module_handle*);		<stdlib.h>
struct lconv localeconv(void);	√	<locale.h>

Function and Parameters	SAA Function?	Header
struct tm localtime(const time_t *timeval*);	√	<time.h>
double log(double *x*);	√	<math.h>
double log1 (double *x*);	√	<math.h>
void longjmp(jmp_buf *env*, int *value*);	√	<setjmp.h>
unsigned long _lrotl(unsigned long *value*, int *shift*);		<stdlib.h>
unsigned long _lrotr(unsigned long *value*, int *shift*);		<stdlib.h>
long int _lseek(int *handle*, long int *offset*, int *origin*);		<io.h> <stdio.h>
char _ltoa(long int *value*, char *string*, int *radix*);		<stdlib.h>
void _makepath(char *path*, char *drive*, char *dir*, char *fname*, char *ext*);		<stdlib.h>
void malloc(size_t *size*);	√	<stdlib.h> <malloc.h>
int _matherr(struct exception *x*);		<math.h>
type max(*type a, type b*);		<stdlib.h>
int mblen(const char *string*, size_t *n*);	√	<stdlib.h>
size_t mbstowcs(wchar_t *pwc*, const char *string*, size_t *n*);	√	<stdlib.h>

Function and Parameters	SAA Function?	Header
int mbtowc(wchar_t *pwc*, const char *string*, size_t *n*);	√	<stdlib.h>
void memccpy(void *dest*, void *src*, int *c*, unsigned *cnt*);		<string.h> <memory.h>
void memchr(const void *buf*, int *c*, size_t *count*);	√	<string.h> <memory.h>
int memcmp(const void *buf1*,const void *buf2*, size_t *count*);	√	<string.h> <memory.h>
void memcpy(void *dest*, const void *src*, size_t *count*);	√	<string.h> <memory.h>
int memicmp(void *buf1*, void *buf2*, unsigned int *cnt*);		<string.h> <memory.h>
void memmove(void *dest*, const void *src*, size_t *count*);	√	<string.h> <memory.h>
void memset(void *dest*, int *c*, size_t *count*);	√	<string.h> <memory.h>
type min(*type a*, *type b*);		<stdlib.h>
int _mkdir(char *pathname*);		<direct.h>
time_t mktime(struct tm *time*);	√	<time.h>
double modf(double *x*, double *intptr*);	√	<math.h>
onexit_t _onexit(onexit_t *func*);		<stdlib.h>
int _open(char *pathname*, int *oflag*, int *pmode*);		<io.h> <fcntl.h> <sys\stat.h>

Function and Parameters	SAA Function?	Header
int _outp(const unsigned int *port*, const int *byte*);		<conio.h> <builtin.h>
unsigned long _outpd(const unsigned int *port*, const unsigned long *dword*);		<conio.h> <builtin.h>
unsigned short _outpw(const unsigned int *port*, const unsigned short *word*);		<conio.h> <builtin.h>
unsigned char __parmdwords(void);		<stdlib.h> <builtin.h>
void perror(const char *string*);	√	<stdio.h>
double pow(double *x*, double *y*);	√	<math.h>
int printf(const char *format-string*, *argument_list*);	√	<stdio.h>
int putc(int *c*, FILE *stream*);	√	<stdio.h>
int putchar(int *c*);	√	<stdio.h>
int _putch(int *c*);		<conio.h>
int _putenv(char *envstring*);		<stdlib.h>
int puts(const char *string*);	√	<stdio.h>
void qsort(void *base*, size_t *num*, size_t *width*, int(*cmpare*)(const void *element1*, const void *element2*));	√	<stdlib.h> <search.h>
int raise(int *sig*);	√	<signal.h>

Function and Parameters	SAA Function?	Header
int rand(void);	√	<stdlib.h>
int _read(int *handle*, char *buffer*, unsigned int *count*);		<io.h>
void realloc(void *ptr*, size_t *size*);	√	<stdlib.h> <malloc.h>
int remove(const char *filename*);	√	<stdio.h>
int rename(const char *oldname*, const char *newname*);	√	<stdio.h>
void rewind(FILE *stream*);	√	<stdio.h>
int _rmdir(char *pathname*);		<direct.h>
int _rmtmp(void);		<stdio.h>
unsigned int _rotl(unsigned int *value*, int *shift*);		<stdlib.h>
unsigned int _rotr(unsigned int *value*, int *shift*);		<stdlib.h>
int scanf(const char *format-string*, *argument_list*);	√	<stdio.h>
void _searchenv(char *name*, char *env_var*, char *path*);		<stdlib.h>
void setbuf(FILE *stream*, char *buffer*);	√	<stdio.h>
int _set_crt_msg_handle(int *handle*);		<stdio.h>
int setjmp(jmp_buf *env*);	√	<setjmp.h>

Function and Parameters	SAA Function?	Header
char setlocale(int *category*, const char *locale*);	√	<locale.h>
int _setmode(int *handle*, int *mode*);		<io.h>
int setvbuf(FILE *stream*, char *buf*, int *type*, size_t *size*);	√	<stdio.h>
void (signal(int *sig*, void (*func*) (int)))(int);	√	<signal.h>
double sin(double *x*);	√	<math.h>
double sinh(double *x*);	√	<math.h>
int _sopen(char *pathname*, int *oflag*, int *shflag*, int *pmode*);		<io.h> <fcntl.h> <share.h> <sys\stat.h>
int _spawnl(int *modeflag*, char *pathname*, char *arg* , char *arg1*, ..., char *argn*, NULL);		<process.h>
int _spawnle(int *modeflag*, char *pathname*, char *arg* , char *arg1*, ..., char *argn*, NULL, char *envp*[]);		<process.h>
int _spawnlp(int *modeflag*, char *pathname*, char *arg* , char *arg1*, ..., char *argn*, NULL);		<process.h>

Function and Parameters	SAA Function?	Header
int _spawnlpe(int *modeflag*, char *pathname*, char *arg* , char *arg1*, ..., char *argn*, NULL, char *envp*[]);		<process.h>
int _spawnv(int *modeflag*, char *pathname*, char *argv*[]);		<process.h>
int _spawnve(int *modeflag*, char *pathname*, char *argv*[], char *envp*[]);		<process.h>
int _spawnvp(int *modeflag*, char *pathname*, char *argv*[]);		<process.h>
int _spawnvpe(int *modeflag*, char *pathname*, char *argv*[], char *envp*[]);		<process.h>
void _splitpath(char *path*, char *drive*, char *dir*, char *fname*, char *ext*);		<stdlib.h>
int sprintf(char *buffer*, const char *format-string*, *argument_list*);	√	<stdio.h>
double sqrt(double *x*);	√	<math.h>
void srand(unsigned int *seed*);	√	<stdlib.h>
int sscanf(const char *buffer*, const char *format*, *argument_list*);	√	<stdio.h>

Function and Parameters	SAA Function?	Header
int _stat(char *pathname*, struct stat *buffer*);		<sys\stat.h> <sys\types.h>
unsigned int _status87(void);		<float.h>
char strcat(char *string1*, const char *string2*);	√	<string.h>
char strchr(const char *string*, int *c*);	√	<string.h>
int strcmp(const char *string1*, const char *string2*);	√	<string.h>
int strcmpi(const char *string1*, const char *string2*);		<string.h>
int strcoll(const char *string1*, const char *string2*);	√	<string.h>
char strcpy(char *string1*, const char *string2*);	√	<string.h>
size_t strcspn(const char *string1*, const char *string2*);	√	<string.h>
char _strdate(char *date*);		<time.h>
char strdup(const char *string*);		<string.h>
char strerror(int *errnum*);	√	<string.h>
char _strerror(char *string*);		<string.h>

Function and Parameters	SAA Function?	Header
size_t strftime(char *dest*, size_t *maxsize*, const char *format*, const struct tm *timeptr*);	√	<time.h>
int stricmp(const char *string1*, const char *string2*);		<string.h>
size_t strlen(const char *string*);	√	<string.h>
char strlwr(char *string*);		<string.h>
char strncat(char *string1*, const char *string2*, size_t *count*);	√	<string.h>
int strncmp(const char *string1*, const char *string2*, size_t *count*);	√	<string.h>
char strncpy(char *string1*, const char *string2*, size_t *count*);	√	<string.h>
int strnicmp(const char *string1*, const char *string2*, size_t *count*);		<string.h>
char strnset(char *string*, int *c*, size_t *count*);		<string.h>
char strpbrk(const char *string1*, const char *string2*);	√	<string.h>
char strrchr(const char *string*, int *c*);	√	<string.h>
char strrev(char *string*);		<string.h>
char strset(char *string*, int *c*);		<string.h>

Function and Parameters	SAA Function?	Header
size_t strspn(const char *string1*, const char *string2*);	√	<string.h>
char strstr(const char *string1*, const char *string2*);	√	<string.h>
char _strtime(char *time*);		<time.h>
double strtod(const char *nptr*, char *endptr*);	√	<stdlib.h>
char strtok(char *string1*, const char *string2*);	√	<string.h>
long int strtol(const char *nptr*, char *endptr*, int *base*);	√	<stdlib.h>
long double strtold(const char *nptr*, char *endptr*);		<stdlib.h>
unsigned long strtoul(const char *nptr*, char *endptr*, int *base*);	√	<stdlib.h>
char strupr(char *string*);		<string.h>
size_t strxfrm(char *string1*, const char *string2*, size_t *count*);	√	<string.h>
void _swab(char *source*, char *destination*, int *n*);		<stdlib.h>
int system(const char *string*);	√	<stdlib.h> <process.h>
double tan(double *x*);	√	<math.h>
double tanh(double *x*);	√	<math.h>

Function and Parameters	SAA Function?	Header
long int _tell(int *handle*);		<io.h>
char _tempnam(char *dir*, char *prefix*);		<stdio.h>
void _threadstore(void);		<stdlib.h>
time_t time(time_t *timeptr*);	√	<time.h>
FILE tmpfile(void);	√	<stdio.h>
char tmpnam(char *string*);	√	<stdio.h>
int _toascii(int *c*);		<ctype.h>
int tolower(int *c*);	√	<ctype.h>
int _tolower(int *c*);		<ctype.h>
int toupper(int *c*);	√	<ctype.h>
int _toupper(int *c*);		<ctype.h>
void _tzset(void);		<time.h>
char _ultoa(unsigned long *value*, char *string*, int *radix*);		<stdlib.h>
int _umask(int *mode*);		<io.h> <sys\stat.h>
int ungetc(int *c*, FILE *stream*);	√	<stdio.h>
int _ungetch(int *c*);		<conio.h>
int _unlink(char *pathname*);		<stdio.h>
int _utime(char *pathname*, struct utimbuf *times*);		<sys\utime.h> <sys\types.h>
var-type va_arg(va_list *arg-ptr*, *var-type*);	√	<stdarg.h>

Function and Parameters	SAA Function?	Header
void va_end(va_list *arg-ptr*);	√	<stdarg.h>
void va_start(va_list *arg-ptr*, *var-name*);	√	<stdarg.h>
int vfprintf(FILE *stream*, const char *format*, va_list *arg-ptr*);	√ √	<stdio.h> <stdarg.h>
int vprintf(const char *format*, va_list *arg-ptr*);	√ √	<stdio.h> <stdarg.h>
int vsprintf(char *target-string*, const char *format*, va_list *arg-ptr*);	√ √	<stdio.h> <stdarg.h>
int _wait(int *stat_loc*);		<process.h>
wchar_t wcscat(wchar_t *string1*, const wchar_t *string2*);	√	<wcstr.h>
wchar_t wcschr(const wchar_t *string1*, wchar_t *character*);	√	<wcstr.h>
int wcscmp(const wchar_t *string1*, const wchar_t *string2*);	√	<wcstr.h>
wchar_t wcscpy(wchar_t *string1*, const wchar_t *string2*);	√	<wcstr.h>
size_t wcscspn(const wchar_t *string1*, const wchar_t *string2*);	√	<wcstr.h>
size_t wcslen(const wchar_t *string*);	√	<wcstr.h>

Function and Parameters	SAA Function?	Header
wchar_t wcsncat(wchar_t *string1*, const wchar_t *string2*, size_t *count*);	√	<wcstr.h>
int wcsncmp(const wchar_t *string1*, const wchar_t *string2*, size_t *count*);	√	<wcstr.h>
wchar_t wcsncpy(wchar_t *string1*, const wchar_t *string2*, size_t *count*);	√	<wcstr.h>
wchar_t wcspbrk(const wchar_t *string1*, const wchar_t *string2*);	√	<wcstr.h>
wchar_t wcsrchr(const wchar_t *string*, wchar_t *character*);	√	<wcstr.h>
size_t wcsspn(const wchar_t *string1*, const wchar_t *string2*);	√	<wcstr.h>
size_t wcstombs(char *dest*, const wchar_t *string*, size_t count);	√	<stdlib.h>
wchar_t wcswcs(const wchar_t *string1*, const wchar_t *string2*);	√	<wcstr.h>
int wctomb(char *string*, wchar_t *character*) ;	√	<stdlib.h>
int _write(int *handle*, const void *buffer*, unsigned int *count*);		<io.h>
double _y (double *x*);	√	<math.h>
double _y1(double *x*);	√	<math.h>
double _yn(int *n*, double *x*);	√	<math.h>

®

Part Number: 61G1441
Program Number: 61G1176
61G1426

Printed in U.S.A.

S61G-1441-