How config LCC compiler (32 bits and 64 bits) into CodeBlocks

Name of tutorial : How config LCC compiler (32 bits and 64 bits) into Code::Blocks on Windows 11 64 bits.

```
Code::Blocks : the best and great free IDE for Windows, Linux and ... Mac OS
```

During first run of CB on Windows, this IDE detect automatically some compilers, or present one list of them pre-configured.
It's very good functionnality, but, sometimes, you must "force" these configurations proposed by default to run correctly.
This tuto describe how configure two compilers on Windows : LCC 32 bits and LCC 64 bits.

How to install LCC 32 bits and LCC 64 bits ?

Note that these compilers presents two different versions beetween 32 or 64 bits version => two executables.

WARNING : LCC is only an compiler C, not C++. You can't generate an executable from source file in C++, strict source C is supported.
And, it will not open source (Jacob Navia control use of it).

You can download it from Internet site : https://lcc-win32.services.net/

Files proposed to download are : "lccwin32.exe" and "lccwin64.exe". You must click on these files to install two versions.
These installers ask about installation's directory, and you can choose, by example, "C:\lcc32" and "C:\lcc64".

Normally, after that, next run of CB detect presence of these compilers and proposed it in list of available compiler in main
menu "Settings" and after submenu "Compiler..." : "LCC Compiler" (CB autodetect only 32 bits version).
If you select this, verify that fields describe next are parametered into CB.

In tab "Toolchain executable", you must find in field "Compiler installation directory" :
C:\lcc32 (subdirectory "\bin" automatically searched after this "top" directory),
and in subtab "Program Files", list next :

compilateur C : lcc.exe
compilateur C++ : lcc.exe
linker for dynamic lib : lcclnk.exe
linker for static lib : lcclib.exe
debugger :
resource compiler : lrc.exe
make program : make.exe

If CB propose different values of fields described below, you can change/force it.

After, you select tab "Search directories", and into each subtab, you write with "add" button, if not searched by default :
to compiler : C:\lcc32\include
to linker : C:\lcc32\lib
to resource compiler : C:\lcc32\include

And, with simply source "hellowworld.c", you can test generation of program into IDE CB, choosing "create new project" in main
windows of CB, and choose "console application" with no source proposed by default, because named "main.c" by default, and choose
compiler "LCC Compiler".
You can select good directory/source with option "add file" after first creation of project into CB.

One time project created, you can generate it with selecting main menu "Build" and choose submenu "Rebuild..." (or CTRL-F11).

If, you apply all of precedent instructions, compile and link of your program must be succeeded.

For version 64 bits of LCC, you must return in main menu "Settings" and after submenu "Compiler..." select "LCC Compiler", then
you must copy before this compiler into CB, and after you must rename this new compiler like (by example) "LCC Compiler (64b)".

After, you must repeat instructions, in tab "Toolchain executable", you must type in field "Compiler installation directory" :
C:\lcc64 (subdirectory "\bin" automatically searched after this "top" directory),
and in subtab "Program Files", list next :
compilateur C : lcc64.exe
compilateur C++ : lcc64.exe
linker for dynamic lib : lcclnk64.exe
linker for static lib : lcclib64.exe
debugger :

resource compiler : lrc.exe
make program : make.exe

If CB propose different values of fields described below, you can change/force it.

After, you select tab "Search directories", and into each subtab, you write with "add" button, if not searched by default :
to compiler : C:\lcc64\include64
to linker : C:\lcc64\lib64
to resource compiler : C:\lcc64\include64

And, with simply source "hellowworld.c", you can test generation of program into IDE CB, choosing "create new project" in main
windows of CB, and choose "console application" with no source proposed by default, because named "main.c" by default, and choose
compiler "LCC Compiler (64b)".
You can select good directory/source with option "add file" after first creation of project into CB.

One time project created, you can generate it with selecting main menu "Build" and choose submenu "Rebuild..." (or CTRL-F11).

If, you apply all of precedent instructions, compile and link of your program must be succeeded.

Pleasure of programming is open for you, your imagination is illimited, at your keyboard ! Enjoy !

PS : source file "hellowworld.c" :

/ *Basic example in language C : hellowworld.c* /

#include <stdio.h>

int main(int argc, char *argv[]) {
/ printf() displays the string inside quotation */
printf("Hello, World!");
return 0;
}

PS2 : LCC/LCC64 compiler can be used directly into command console of Windows (CMD.EXE) and next command lines configure it :

SET PATHSAV=%PATH%
REM Choose if you use major directory of LCC/LCC64 beetween 32 bits or 64 bits by select 32 or 64 in next command

SET LCC=C:\lcc{32|64}
SET PATH=%LCC%\bin;%PATH%

By example, to generate 64 bits version of "console" executable :

REM No "one pass" available to generate target with this compiler. Only two passes are supported :
lcc64 hellowworld.c -o hellowworld.obj -IC:\lcc64\include64
lcclnk64 hellowworld.obj -o hellowworld.exe -LC:\lcc64\lib64

WARNING : This "simple" compiler don't create automatically directories or don't interpret variable like "%var%" by example.

After, work with LCC/LCC64 compiler, but, at the end of your work, think to return in initial state ... to avoid difficulties :

.....
SET PATH=%PATHSAV%

PS3 : syntax and options of command "lcc" (same with "lcc64") :

Option Meaning
-A All warnings will be active.
-ansic Disallow the language extensions of lcc-win32.
-ansic90 Conform to ANSI 1990 standard. This option will disallow all C99
and link with crtdll.dll, the C90 standard linbrary. Long long
support doesn't exist, there is no optimizer, but programs are
30K smaller than C99 programs. Use the driver "lc.exe" for this
option. Note that // comments are NOT recognized if you use this
option.
-D Define the symbol following the D. Example:-DNODEBUG The symbol
NODEBUG is defined. Note: NO space between the D and the symbol
-dd Debug defines. This option will write in the standard output a list
of all preprocessor defines followed by the file name and line number
where they are defined. This allows to see which definition is used.
-check Check the given source for errors. No object file is generated.
-E Generate an intermediate file with the output of the preprocessor.
The output file name will be deduced from the input file name, i.e.,
for a compilation of foo.c you will obtain foo.i.
-E+ Like the -E option, but instead of generating a `#line` xxx directive,
the preprocessor generates a # xxx directive.
-EP Like the -E option, but no `#line` directives will be generated.
-errout= Append the warning/error messages to the indicated file.

Example -errout=Myexe.err. This will append to Myexe.err all
warnings and error messages.
-eN Set the maximum error count to N. Example:-e25.
-fno-inline The inline directive is ignored.
-Fo This forces the name of the output file.
-g2 Generate the debugging information.
-g3 Arrange for function stack tracing. If a trap occurs, the function
stack will be displayed.
-g4 Arrange for function stack and line number tracing.
-g5 Arrange for function stack, line number, and return call stack
corruption tracing.
-I Add a path to the path included, i.e. to the path the compiler follows
to find the header files. Example:-Ic:\project\headers. Note NO space
beetween the I and the following path.
-libcdll Use declarations for lcclibc.dll.Files compiled with this option
should use the -dynamic option of the linker lcclnk.
-M Print in standard output the names of all files that the preprocessor
has opened when processing the given input file. No object file is
generated.
-M1 Print in standard output each include file recursively, indicating
where it is called from, and when it is closed.
-nw No warnings will be emitted. Errors will be still printed.
-O Optimize the output. This activates the peephole optimizer.
-o Use as the name of the output file the given name. Identical to the
Fo flag above.
-overflowcheck
Generate code to test for overflow for all additions, subtractions and
multiplications.
-p6 Enable SSE3 instructions
-p6=no Disable SSE3 instructions
-profile Inject code into the generated program to measure execution time.
This option is incompatible with debug level higher than 2.
-S Generate an assembly file. The output file name will be deduced from
the input file: for a compilation of foo.c you will obtain foo.asm.
-s n Set the switch density to n that must be a value between 0.0 and 1.0.
Example: -s 0.1
-shadows Warn when a local variable shadows a global one.
-stackinit n
Initialize the uninitialized stack at function entry with a value "n".
-U Undefine the symbol following the U.

-unused Warns about unused assignments and suppresses the dead code.
-v Show compiler version and compilation date
-z Generate a file with the intermediate language of lcc. The name of the
generated file will have a .lil extension.
-Zp[1,2,4,8,16] Set the default alignment in structures to one, two, four, etc.
If you set it to one, this actually means no alignment at all.

Command line options are case sensitive. Use either "-" or "/" to
introduce an option in the command line.
The compiler detects the type of machine used and will set the p6 flag automatically. Use
the p6 flags in case you want to override detection.

Syntax and options of command "lcclnk" (same with "lcclnk64") :

Lcclnk command line options. Options are introduced with the character '-' or the character '/'.
-o
Sets the name of the output file to file name. Insert a space between the o and the
name of the file.
-errout
Write all warnings/error messages to the indicated file name.
-subsystem
Indicate the type of output file. Console or windows application
-stack-commit
With this option, you can commit more pages than one (4096 bytes).
-stack-reserve
The default stack size is 1MB. This changes this limit.
-dynamic
Use lcclibc.dll as default library and link to it dynamically instead of linking to libc.lib.
-dll
This option indicates to the linker that a .dll should be created instead of an .exe.
-map
Indicates the name of the map file.This option is incompatible with the -s option.
-nolibc
Do not include the standard C library.
-s Strips all symbolic and debugging information from the executable.
-version nn.nn
Adds the version number to the executable.
-errout=
Write all warnings or errors to the file name indicated.Note that no spaces

should separate the name from the equals sign.
-nounderscores
When creating a DLL for Visual Basic for instance, it is better to export names without underscores from a DLL.
-entry
Option fo setting the DLL entry point to the given function
-version
Print the version name

Just to illustrate, here commands to generate an console application with source file C and resource file :

```
lcc64 -g2 -DDEBUG -D_DEBUG -IC:\lcc64\include64 -
FoobjIcc64\Release%NAME_APPLI%.obj src%NAME_APPLI%.c
lrc -IC:\lcc64\include64 -FoobjIcc64\Release%NAME_APPLI%.res src%NAME_APPLI%.rc
lcclnk64 -subsystem console -LC:\lcc64\lib64 objIcc64\Release%NAME_APPLI%.obj
objIcc64\Release%NAME_APPLI%.res -o binIcc64\Release%NAME_APPLI%.exe
glu32.lib opengl32.lib gdi32.lib advapi32.lib comdlg32.lib winmm.lib user32.lib kernel32.lib
```

And, to terminate, an example of generation of DLL with LCC/LCC64 with source file C and resource file :

```
lcc64 -g2 -DDEBUG -D_DEBUG -IC:\lcc64\include64 -FoobjIcc64\Debug%NAME_APPLI%.obj
src%NAME_APPLI%.c
lrc -IC:\lcc64\include64 -FoobjIcc64\Debug%NAME_APPLI%.res src%NAME_APPLI%.rc
lcclnk64 -dll -LC:\lcc64\lib64 objIcc64\Debug%NAME_APPLI%.obj
objIcc64\Debug%NAME_APPLI%.res -o binIcc64\Debug%NAME_APPLI%.exe
glu32.lib opengl32.lib gdi32.lib advapi32.lib comdlg32.lib winmm.lib user32.lib kernel32.lib
lcclib64 /out:binIcc64\Debug%NAME_APPLI%.lib objIcc64\Debug%NAME_APPLI%.obj
```