How config MinGW GCC C_C++ compiler (official version) (32 bits) into CodeBlocks

Full name of tutorial : How config MinGW GCC C/C++ compiler "official version" (32 bits) into Code::Blocks on Windows 11 64 bits.

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

Presentation of MInGW and GCC compiler (if needed ...)

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 very "basic" compiler on Windows : compiler GCC included in package MinGW, official version.

MinGW : A native Windows port of the GNU Compiler Collection (GCC), with freely distributable import libraries and header
files for building native Windows applications; includes extensions to the MSVC runtime to support C99 functionality.
All of MinGW's software will execute on the 64bit Windows platforms.

Remark : MinGW "official version" is only an version 32 bits. Initials authors refuse "porting" to "true" version 64 bits.
I don't know why, but I regret this, and it's reason of creation of "fork" : MINGW64, much up to date and "true" 64 bits.

How to install GNU GCC Compiler C/C++ included in package Mingw32 "official version" (version gcc 9.2.0, very old today ... recent version of gcc is 14.2.0 ... dated 01/08/2024 ...) ?

You can download it from Internet site of Sourceforge :
https://sourceforge.net/projects/mingw/files/Installer/mingw-get-setup.exe
and click to this executable.

With these tool, you must install all software mandatory to develop : "make", "gcc", "binutils", "win32-api", etc...
Your installation of MingW32 "official" is, by default, on directory : C:\MingW.

Configuration of MInGW "official" and GCC compiler into CB

Normally, CB can detect this installation.
But, if not, you can configure this compiler into CB by selecting main menu "Settings" then submenu "Compiler..." into IDE interface.

After, you choose an compiler proposed : by example, "GNU GCC Compiler (default)", and you choose tab" "Toolchain executable" :
C:\MinGW (subdirectory "\bin" automatically searched after this "top" directory)
- compilateur C : gcc.exe
- compilateur C++ : g++.exe
- linker for dynamic lib : g++.exe
- linker for static lib : ar.exe
- debugger : gdb.exe
- resource compiler : windres.exe
- make program : mingw32-make.exe

If CB don't propose different values of fields described below, you can change it.

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

Then, you can type this into command console Windows : "C:\MinGW\bin\gcc.exe --version"; text after must appear :

gcc.exe (MinGW.org GCC Build-2) 9.2.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Test of "simple" code with GCC compiler C/C++ of MinGW "official" into CB

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 "GNU GCC Compiler (default)".
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), save your project CB.

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 : Use of GCC compiler included in MinGW official in command line (just to illustrate)

You can also use compiler GCC in command console on Windows (CMD.EXE) with next instructions :

```
set PATHSAV=%PATH%
set PATH=C:\MinGW\bin;%PATH%
REM Next instructions are not mandatory, but you can set var if you want.
REM set CLANG=C:\MinGW\include
REM set LIBRARY_PATH=C:\MinGW\lib

REM Generate console application in one pass
gcc hellowworld.c -o hellowworld.exe
REM Generate console application in two pass
```

gcc -c hellowworld.c -o hellowworld.obj
gcc hellowworld.obj -o hellowworld.exe -Wl,--subsystem console

Continue with use of GCC, and don't forgive, at the end of your work, to return to initial state :

set PATH=%PATHSAV%

But, it's much easy to use GCC of MinGW directly into CB IDE especially with complex C program (many C sources
and many subdirectories ...) -)

PS3 : Principal syntax of tools GCC

Command "gcc" present a very "verbose" list of options, but to resume principal and useful options, you can use
by example :
"-m16" Generate i386 16 bits object or executable.
"-m32" Generate i386 32 bits object or executable.
"-m64" Generate x64 64 bits object or executable.
"-c" Compile and assemble, but do not link.
"-D var[=value]" Define variable to be use by préprocessor, and optionnally affect an value at this variable.
"-o " Place the output into .
"-I " Add directory to search include files
"-L " Add directory to search library files
"-shared" Create a shared library.
"-llibrary" Give name of library used by linker.
"-pthread" Link with the POSIX threads library.
"-Wl," Pass comma-separated on to the linker. Example of options :
-t trace all input files used by linker
--kill-at not add the function name decorations at-sign and number for stdcall functions
--add-stdcall-alias export functions with the stdcall decoration suffix (@nn) and also without this suffix
--dll create DLL on Win32 systems
--export-all-symbols all global symbols in the objects used to build a DLL will be exported by the DLL
--output-def file create a def file of all exported sysmbols
--out-implib file create a import library in parallel of creation of shared library
--subsystem ident create target based on "ident" that can be valued by "native, windows, console, posix, or xbox"

You can consult an summary of these options on site :
https://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html
And for "ld" (linker og gcc), you can consult too : http://sourceware.org/binutils/docs-2.16/ld/Options.html