How config CYGWIN64 (+ MinGW32 or MinGW64) into CodeBlocks

Name of tutorial : How config CYGWIN64 (+ MinGW32/MinGW64) 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 compilers GCC included in package CYGWIN + MinGW32 and MinGW64 into Code::Blocks on Windows 11
64 bits.

But, before describe how configure CYGWIN64 into CB, it's seems necessary to explain what is CYGWIN64, and also what sort of compilers GCC are configurable into CB.

On Windows 11 systems 64 bits, you can't install 32 bits version of CYGWIN, only 64 bits version (CYGWIN64).
The installation binary provide by CYGWIN team is available on this site https://cygwin.com/install.html, and is named : "setup-x86_64.exe".
By default, the install directory is "C:\cygwin64", and, since more years, you can upgrade with the same tool to integrate MinGW32/64 directly into CYGWIN environment.
Think to install too "make" (GNU) tool to "cmake" generator "Unix Makefiles"
by example, it's not installed by default ...

REMARK : CYGWIN is not the only implementation of Linux/Unix layers on Windows systems.
WSL (Windows Subsystem Linux) is the second,
provided by Microsoft with agreement of Linux distributers supported (ex : Ubuntu 22.04, Ubuntu 24.04, Kali Linux, etc...)
(I don't forget emulation/virtualization on Windows that allow also to run directly Linux/Unix into "virtual environment").

It's seems also important to understand some difference beetween use of three GCC compilers available into CYGWIN64 :
- "native" GCC compiler 64 bits use to construct "unix" like application with POSIX layer, but also pure Windows application,

- GCC compiler 32 bits included in MinGW32 (version available on "official" repository of CYGWIN) to construct pure Windows
application,
- GCC compiler 64 bits included in MinGW64 (version available on "official" repository of CYGWIN) to construct pure Windows
application,

The names of executable of these three compilers available "into" CYGWIN64 are :
- gcc.exe or x86_64-w64-pc-gcc.exe (version native de GCC) 12.4.0 (64 bits), InstalledDir : C:/Cygwin64/bin, ident that can be used
by preprocessing : **CYGWIN**
- i686-w64-mingw32-gcc (GCC) 12.4.0 (32 bits), InstalledDir : C:/Cygwin64/bin, ident that can be used by preprocessing : **MINGW32**
(and presence of copy of this executable : i686-w64-mingw32-gcc-12)
- x86_64-w64-mingw32-gcc (GCC) 12.4.0 (64 bits), InstalledDir : C:/Cygwin64/bin, ident that can be used by preprocessing : **MINGW64**
(and presence of copy of this executable : x86_64-w64-mingw32-gcc-12)

NB : Two versions of MinGW32/64 GCC compilers define also variable _WIN32 that can be used into preprocessing, but "native" GCC not ... and it's normal !!!

Just to be precise, you can search on Internet a discution about this :
"Dependency of dll "cygwin1.dll" after construct program with CYGWIN environment" (here version 64 bits)

Internet research about dependency with dll "cygwin1.dll" indicate this first comment :

"Cygwin is a Linux-like environment for Windows. It consists of a DLL ("cygwin1.dll"), which acts as an emulation layer providing
substantial POSIX (Portable Operating System Interface) system call functionality, and a collection of tools, which provide a Linux
look and feel."

"A basic "Hello, World" Cygwin program requires two library to run (.ie. program generated by "gcc" (same "x86_64-pc-cygwin_gcc.exe")
or "i686-pc-cygwin-gcc.exe" if you install 32 bits version of Cygwin on Windows 32 bits) :
a) the main Cygwin DLL "cygwin1.dll", and
b) cyggcc_s-1.dll (dll special GCC).
Into Cygwin terminal, "cygwin1.dll" is present on directory /bin.
MinGW program into CYGWIN will be linked to msvcrt.dll which is an internal, undocumented, unversioned library that is part of Windows,
and off-limits to application use. That library is essentially a fork of the redistributable run-time

library from MS Visual C for use
by Windows itself."

Invert "/" by "" separator of directories's level, if use research of these DLL in console command Windows, instead a cygwin terminal.

Before beginning, think to add C:\Cygwin64\bin at your PATH Windows, to access at the binaries of compilers and needed tools ("make",
"file" and "ldd" tools by example).

Proof of concept : search "all" dependency of executable generated by all GCC compilers (three in all) included in CYGWIN64.

One simple example to confirm that (or not !!!) (One Cygwin terminal open, and console command Windows too, positioned in same directory
(that contains source and executables)) :

Source C in file "hello_sample.c" :

*** Begin file "hello_sample.c" **

```
#include <stdio.h>

int main(){
printf("hello world\n");
return 0;
}
```

*** End file "hello_sample.c" **

After, execute these next commands in Cygwin terminal :

"$ gcc.exe -Wall hello_sample.c -o hello_x86_64-pc-cygwin.exe
$ ./hello_x86_64-pc-cygwin.exe
hello world
$ file hello-x86_64-pc-cygwin.exe
hello.exe: PE32+ executable (console) x86-64, for MS Windows, 18 sections
$ ldd hello_x86_64-pc-cygwin.exe
ntdll.dll => /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll (0x7ff802ab0000)
KERNEL32.DLL => /cygdrive/c/WINDOWS/System32/KERNEL32.DLL (0x7ff801c80000)
KERNELBASE.dll => /cygdrive/c/WINDOWS/System32/KERNELBASE.dll (0x7fffffc40000)
cygwin1.dll => /usr/bin/cygwin1.dll (0x7ffec9370000)

(linked with cygwin1.dll, run in console command windows fail :
"Impossible to run the code, because cygwin1.dll is undiscovered"

To resolve, copy cygwin1.dll in directory C:\Windows\system32
or add C:\Cygwin64\bin to your Windows PATH. Linked too with ntdll.dll
and kernel32.dll. After that, tests OK with both cygwin terminal and
console command windows)

$ i686-w64-mingw32-gcc.exe -Wall hello_sample.c -o hello_i686-w64-mingw32.exe
$ ./hello_i686-w64-mingw32.exe
hello world
$ file hello_i686-w64-mingw32.exe
hello.exe: PE32 executable (console) Intel 80386, for MS Windows, 18 sections
$ ldd hello_i686-w64-mingw32.exe
ntdll.dll => /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll (0x7ff802ab0000)
ntdll.dll => /cygdrive/c/Windows/SysWOW64/ntdll.dll (0x779f0000)
wow64.dll => /cygdrive/c/WINDOWS/System32/wow64.dll (0x7ff801e30000)
wow64base.dll => /cygdrive/c/WINDOWS/System32/wow64base.dll (0x7ff801bf0000)
wow64win.dll => /cygdrive/c/WINDOWS/System32/wow64win.dll (0x7ff802060000)
wow64con.dll => /cygdrive/c/WINDOWS/System32/wow64con.dll (0x7ff801bd0000)

(not linked with cygwin1.dll, it's normal, but not linked too with msvcrt.dll.
However, linked with ntdll.dll (64 bits, and 32 bits version) and wow64.dll and
derivative of this DLL.
Tests OK with both cygwin terminal and console command windows)

$ x86_64-w64-mingw32-gcc.exe -Wall hello_sample.c -o hello_x86_64-w64-mingw32.exe
$ ./hello_x86_64-w64-mingw32.exe
hello world
$ file hello_x86_64-w64-mingw32.exe
hello.exe: PE32+ executable (console) x86-64, for MS Windows, 20 sections
$ ldd hello_x86_64-w64-mingw32.exe
ntdll.dll => /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll (0x7ff802ab0000)
KERNEL32.DLL => /cygdrive/c/WINDOWS/System32/KERNEL32.DLL (0x7ff801c80000)
KERNELBASE.dll => /cygdrive/c/WINDOWS/System32/KERNELBASE.dll (0x7fffffc40000)
msvcrt.dll => /cygdrive/c/WINDOWS/System32/msvcrt.dll (0x7ff8021f0000)

(not linked with cygwin1.dll, it's normal. However, linked with
ntdll.dll and kernel32.dll AND msvcrt.dll.
Tests OK with both cygwin terminal and console command windows)

Conclusion to detect presence of DLL "cygwin1.dll" into executable, with last version of
CYGWIN 64 bits
(version DLL cygwin1.dll : 3.5.3) :

```
a) nothing reference to DLL "cyggcc_s-1.dll" with all version of GCC compiler
used ("native" or MinGW32/64)
        (old version of GCC ?)
b) with all version of GCC compiler used, always linked with "main" DLL of
Windows 64 bits "ntdll.dll" present on
        directory C:\Windows\System32,
c) with version 32 bits of GCC compiler (MinGW32), linked too with DLL 32 bits
of Windows "ntdll.dll" present on
        directory C:\Windows\SysWOW64, and DLL "wow64.dll" and it's
derivative, presents on directory  C:\Windows\System32,
d) with version 64 bits of two GCC compilers ("native" and MinGW64), linked
with "kernel32.dll" and "kernelbase.dll"
        presents on directory  C:\Windows\System32,
e) with only version MinGW64 of GCC compiler, linked with "msvcrt.dll",
present on directory C:\Windows\System32,
f) with only version "native" of GCC compiler, linked with "cygwin1.dll",
present on directory /usr/bin/ (or C:\Cygwin64\bin).
```

Conclusion : Dependency of dll "cygwin1.dll" is only implemented with "native" version of GCC compiler into Cygwin 64 bits on
Windows 64 bits, not with MinGW32/64 versions of GCC compiler. And, to assure portability on Windows, use of many "system" DLL
present on system directory with mix beetween direct 64 bits DLL and "derivated" 32 bits DLL used by layer "WoW".

Consequence : Only DLL/executable generated by "native" GCC compiler need presence of DLL "cygwin1.dll" in the PATH, or directly
in same directory of this DLL/executable.

After this long explanation (sorry !), return now about how configure these three compilers into CB.

You must configure this new compiler into CB by selecting main menu "Settings" then submenu "Compiler..." into IDE interface,
then, choose "GNU GCC Compiler (default)" and rename this like "CYGWIN native GCC 64 bits" by example, if it's not detected.

Then, you choose tab "Toolchain executable" to position good environment like this

Toolchain executable :
C:\CYGWIN64 (subdirectory \bin will be searched automatically to access to binaries listed

after)
compilateur C : gcc.exe
compilateur C++ : g++.exe
linker for dynamic lib : g++.exe
linker for static lib : gcc-ar.exe
debugger : GDB (default)
resource compiler : windres.exe
make program : make.exe (remenber that, this tool is not installed by default)

Then, you choose tab" "Search directories" to position good environment like this :

```
to compiler :                   C:\CYGWIN64\usr\include   and
C:\CYGWIN64\usr\include\w32api (if you want used windows API)
to linker :                     C:\CYGWIN64\lib     and
C:\CYGWIN64\lib\w32api             (if you want used windows API)
                                                (but you can add also
C:\CYGWIN64\lib\X11 and C:\CYGWIN64\lib\xorg, if you want used X GUI API)
to resource compiler :  C:\CYGWIN64\usr\include   and
C:\CYGWIN64\usr\include\w32api (if you want used windows API)
```

Then, you can type this into command console Windows : "C:\CYGWIN64\bin\g++.exe --version"; result here :

g++ (GCC) 12.4.0
Copyright (C) 2022 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.

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", and choose compiler
"CYGWIN native GCC 64 bits".
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 with this new compiler must be succeeded.

For CYGWIN64 MinGW32, you must return into menu "Settings" of CB and select "CYGWIN native GCC 64 bits" compiler for copy it first,

and, to terminate, rename to "CYGWIN MinGW32 GCC" by example.

Then, you choose tab "Toolchain executable" of this compiler to position good environment like this

Toolchain executable :
C:\CYGWIN64 (subdirectory \bin will be searched automatically to access to binaries listed after)
compilateur C : i686-w64-mingw32-gcc.exe
compilateur C++ : i686-w64-mingw32-g++.exe
linker for dynamic lib : i686-w64-mingw32-g++.exe
linker for static lib : i686-w64-mingw32-gcc-ar.exe
debugger : GDB (default)
resource compiler : i686-w64-mingw32-windres.exe
make program : make.exe (note that mingw32-make.exe is not present with this install)

You must add another directory into tab "Toolchain executable" with subtab "Additional paths" : C:\TDM-GCC-32\gdb32\bin.
But, the name of executable associated with gdb is "gdb32.exe" not "gdb.exe" and you can't change this name into interface CB.
But, to resolve this, you can copy, in this directory, "gdb32.exe" to "gdb.exe", and "gdb32server.exe" to "gdbserver.exe".
Or create two links pointed on these two executables.

Then, you choose tab" "Search directories" to position good environment like this :

```
to compiler :                    C:\CYGWIN64\usr\i686-w64-mingw32\sys-
root\mingw\include
to linker :                      C:\cygwin64\usr\i686-w64-mingw32\sys-
root\mingw\lib
to resource compiler :  C:\CYGWIN64\usr\i686-w64-mingw32\sys-
root\mingw\include
```

You can test with precedent project C++ create into CB, after change compiler to "CYGWIN MinGW32 GCC" and rebuild it.

And finally, for CYGWIN64 MinGW64, you must return into menu "Settings" of CB and select "CYGWIN MinGW32 GCC" compiler for copy
it first, and, to terminate, rename to "CYGWIN MinGW64 GCC" by example.

Then, you choose tab "Toolchain executable" of this compiler to position good environment like this

Toolchain executable :
C:\CYGWIN64 (subdirectory \bin will be searched automatically to access to binaries listed after)
compilateur C : x86_64-w64-mingw32-gcc.exe
compilateur C++ : x86_64-w64-mingw32-g++.exe
linker for dynamic lib : x86_64-w64-mingw32-g++.exe
linker for static lib : x86_64-w64-mingw32-gcc-ar.exe
debugger : GDB (default)
resource compiler : x86_64-w64-mingw32-windres.exe
make program : make.exe (note that mingw32-make.exe is not present with this install)

Then, you choose tab" "Search directories" to position good environment like this :

```
to compiler :                    C:\CYGWIN64\usr\x86_64-w64-mingw32\sys-
root\mingw\include
to linker :                      C:\cygwin64\usr\x86_64-w64-mingw32\sys-
root\mingw\lib
to resource compiler :  C:\CYGWIN64\usr\x86_64-w64-mingw32\sys-
root\mingw\include
```

You can test with precedent project C++ create into CB, after change compiler to "CYGWIN MinGW64 GCC" and rebuild it.

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 : You can also use three compilers GCC included in CYGWIN64 in command console on Windows (CMD.EXE) with next instructions :

```
set PATHSAV=%PATH%
set PATH=C:\CYGWIN64\bin;%PATH%

REM First with "native" GCC of CYGWIN64

REM Generate console application in one pass
gcc hellowworld.c -o hellowworld.exe
REM Generate console application in two pass, compile only first, and link to terminate.
gcc -c hellowworld.c -o hellowworld.obj
gcc hellowworld.obj -o hellowworld.exe -Wl,--subsystem=console

REM Second with MinGW32 GCC of CYGWIN64

REM Generate console application in one pass
i686-w64-mingw32-gcc.exe -m32 hellowworld.c -o hellowworld.exe -I C:\CYGWIN64\usr\i686-
w64-mingw32\sys-root\mingw\include
-L C:\cygwin64\usr\i686-w64-mingw32\sys-root\mingw\lib
REM Generate console application in two pass, compile only first, and link to terminate.
i686-w64-mingw32-gcc.exe -m32 -c hellowworld.c -o hellowworld.obj -I C:\CYGWIN64\usr\i686-
w64-mingw32\sys-root\mingw\include
i686-w64-mingw32-gcc.exe -m32 hellowworld.obj -o hellowworld.exe -Wl,--subsystem=console
-L C:\cygwin64\usr\i686-w64-mingw32\sys-root\mingw\lib

REM Third with MinGW64 GCC of CYGWIN64

REM Generate console application in one pass
x86_64-w64-mingw32-gcc.exe -m64 hellowworld.c -o hellowworld.exe -I
C:\CYGWIN64\usr\x86_64-w64-mingw32\sys-root\mingw\include
-L C:\cygwin64\usr\x86_64-w64-mingw32\sys-root\mingw\lib
REM Generate console application in two pass, compile only first, and link to terminate.
x86_64-w64-mingw32-gcc.exe -m64 -c hellowworld.c -o hellowworld.obj -I
C:\CYGWIN64\usr\x86_64-w64-mingw32\sys-root\mingw\include
x86_64-w64-mingw32-gcc.exe -m64 hellowworld.obj -o hellowworld.exe -Wl,--
subsystem=console -L C:\cygwin64\usr\x86_64-w64-mingw32\sys-root\mingw\lib
```

Continue with use of CYGWIN64 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 TDM GCC MinGW32 or TDM GCC MinGW64 directly into CB IDE especially with complex C program
(many C sources and many subdirectories ...) -)

PS3 : 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.
"-mwindows" Create GUI application on Win32 systems (entry point of program "WinMain", not "main")
"-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"

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