維基教科書
自由世界　自由讀本

TCC使用指南

維基教科書，自由的教學讀本

本頁面介紹的是 Tiny C Compiler 的使用資訊。

程式安裝

- Windows環境安裝，設定與測試

將win32壓縮檔案0.9.26-win32位元版本 (http://download.savannah.nongnu.org/releases/tinycc/tcc-0.9.26-win32-bin.zip)解壓縮於任意目錄下皆可以（其他版本0.9.26-win64位元版本 (http://download.savannah.gnu.org/releases/tinycc/tcc-0.9.26-win64-bin.zip)，0.9.26 Linux版本與原始碼檔 (http://download.savannah.gnu.org/releases/tinycc/tcc-0.9.26.tar.bz2)），這裏是示範於windows系統C磁碟下範例

```
C:\TCC或C:\>MD TCC<enter>
```

設定系統環境參數，新增

```
變數名稱：TCC
參數為：C:\TCC
```

增加路徑

```
變數名稱：path
參數：{原有的路徑參數};%TCC%;
```

測試：開啟命令提示字元（Command Prompt）於任一目錄下輸入TCC -version，即顯示如下

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\TCC -version <enter>
tcc version 0.9.26 (i386 Win32)
```

或

```
C:\TCC -version <enter>
tcc version 0.9.27 (x86_64 Windows)
```

往後即可於任一目錄下編譯C語言程式碼

使用方式

- 可以相同於一般的命令列C語言軟體的使用方式

- 編譯方式

於Windows下編譯

```
C:\>tcc <filename.c>
```

或是

```
C:\>tcc -run <filename.c>
```

提示：以"-run"方式來編譯程式，編譯結果祇會存放於記憶中而已，執行完畢後即釋放，而不會產生對應的執行檔（即EXE）。

於Linux/Unix Like下編譯

```
/usr/local/bin/tcc <filename.c>
```

或是

```
/usr/local/bin/tcc -run <filename.c>
```

同前述"-run"的提示。

指令查詢

可於DOS/Windows命令提示字元下，或是於Linux/Unix Like下鍵入

```
 C:\>TCC <enter>或是[folder name]$ TCC <center>
```

64位元版本即會得到如下說明語法：

```
C:\tcc -version <enter>
tcc version 0.9.26 (x86-64 Win64)

C:\tcc <enter>
tcc version 0.9.26 - Tiny C Compiler - Copyright (C) 2001-2006 Fabrice Bellard
Usage: tcc [options...] [-o outfile] [-c] infile(s)...
       tcc [options...] -run infile [arguments...]
General options:
  -c           compile only - generate an object file
  -o outfile   set output filename
  -run         run compiled source
  -fflag       set or reset (with 'no-' prefix) 'flag' (see man page)
  -Wwarning    set or reset (with 'no-' prefix) 'warning' (see man page)
  -w           disable all warnings
  -v           show version
  -vv          show included files (as sole argument: show search paths)
  -dumpversion
  -bench       show compilation statistics
Preprocessor options:
  -E           preprocess only
  -Idir        add include path 'dir'
  -Dsym[=val] define 'sym' with value 'val'
  -Usym        undefine 'sym'
Linker options:
  -Ldir        add library path 'dir'
```

```
  -llib       link with dynamic or static library 'lib'
  -pthread    link with -lpthread and -D_REENTRANT (POSIX Linux)
  -r          generate (relocatable) object file
  -rdynamic   export all global symbols to dynamic linker
  -shared     generate a shared library
  -soname     set name for shared library to be used at runtime
  -static     static linking
  -Wl,-opt[=val]  set linker option (see manual)
Debugger options:
  -g          generate runtime debug info
  -b          compile with built-in memory and bounds checker (implies -g)
  -bt N       show N callers in stack traces
Misc options:
  -nostdinc   do not use standard system include paths
  -nostdlib   do not link with standard crt and libraries
  -Bdir       use 'dir' as tcc internal library and include path
  -MD         generate target dependencies for make
  -MF depfile put generated dependencies here
```

64位元版本即會得到如下說明語法：

```
C:\Program Files\tcc>tcc
Tiny C Compiler 0.9.27 - Copyright (C) 2001-2006 Fabrice Bellard
Usage: tcc [options...] [-o outfile] [-c] infile(s)...
       tcc [options...] -run infile [arguments...]
General options:
  -c          compile only - generate an object file
  -o outfile  set output filename
  -run        run compiled source
  -fflag      set or reset (with 'no-' prefix) 'flag' (see tcc -hh)
  -Wwarning   set or reset (with 'no-' prefix) 'warning' (see tcc -hh)
  -w          disable all warnings
  -v -vv      show version, show search paths or loaded files
  -h -hh      show this, show more help
  -bench      show compilation statistics
  -           use stdin pipe as infile
  @listfile   read arguments from listfile
Preprocessor options:
  -Idir       add include path 'dir'
  -Dsym[=val] define 'sym' with value 'val'
  -Usym       undefine 'sym'
  -E          preprocess only
Linker options:
  -Ldir       add library path 'dir'
  -llib       link with dynamic or static library 'lib'
  -r          generate (relocatable) object file
  -shared     generate a shared library/dll
  -rdynamic   export all global symbols to dynamic linker
  -soname     set name for shared library to be used at runtime
  -Wl,-opt[=val]  set linker option (see tcc -hh)
Debugger options:
  -g          generate runtime debug info
  -b          compile with built-in memory and bounds checker (implies -g)
  -bt N       show N callers in stack traces
Misc. options:
  -x[c|a|n]   specify type of the next infile
  -nostdinc   do not use standard system include paths
  -nostdlib   do not link with standard crt and libraries
  -Bdir       set tcc's private include/library dir
  -MD         generate dependency file for make
  -MF file    specify dependency file name
  -m32/64     defer to i386/x86_64 cross compiler
Tools:
  create library  : tcc -ar [rcsv] lib.a files
  create def file : tcc -impdef lib.dll [-v] [-o lib.def]
```

檔案大小

網路上Demon's Blog (http://demon.tw/software/tiny-c-compiler.html)亦有測試資訊，以Borland C Compiler 5.5 (https://en.wikipedia.org/wiki/Borland_C%2B%2B) （ BCC, Command-line, Freeware ）, Visual C++ 6.0與Tiny C Compiler來比較編譯後的檔案大小。

編譯結果為：

- 用Borland C Compiler 5.5編譯結果為 51.0 KB（ 52,224 bytes ）
- 用Visual C++ 6.0編譯結果為 40.0 KB（ 40,960 bytes ）
- 用TCC 0.9.25(win32)/0.9.26(win32)編譯結果為 1.50 KB（ 1,536 bytes ）
- 用TCC 0.9.26(win64)/0.9.27(win64)編譯結果為 2.0 KB（ 2,048 bytes ）

程式編譯

基本編譯

測試編譯程式碼為：

```
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("Hello, world\n");
    return 0;
}
```

存成檔案"hello.c"，接著編譯程式

```
C:\tcc hello.c <enter>
```

若沒有其他資訊，則是編譯完成，接著執行程式

```
C:\hello <enter>
Hello, world!
```

或是於編譯（32位元）時增加參數，有多的資訊可以參考如下：

```
C:\tcc -v -bench hellow.c <enter>
```

32位元編譯時資訊如下：

```
tcc version 0.9.26 (i386 Win32)
-> hellow.c
1245 idents, 1235 lines, 48252 bytes, 0.001 s, 1234999 lines/s, 48.3 MB/s
<- hellow.exe (1536 bytes)
```

64位元編譯時資訊如下：

```
tcc version 0.9.26 (x86-64 Win64)
-> hellow.c
1275 idents, 1234 lines, 48241 bytes, 0.001 s, 1234000 lines/s, 48.2 MB/s
<- hellow.exe (2048 bytes)
```

(新版)64位元編譯時資訊如下：

```
tcc version 0.9.27 (x86_64 Windows)
-> hellow.c
<- hellow.exe (5120 bytes)
* 20240 idents, 24604 lines, 906348 bytes
* 0.031 s, 793677 lines/s, 29.2 MB/s
```

編譯測試

以有錯誤的程式碼測試如下：

```
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("Hello, world\n);
}
```

測試編譯時，結果如下：

```
tcc version 0.9.26 (i386 Win32)
-> hellow.c
hellow.c:5: error: missing terminating " character
```

記憶體編譯

如使用記憶體內編譯（32位元）方式時，方法與結果如下：

```
C:\tcc -v -bench -run hellow.c <enter>
tcc version 0.9.26 (i386 Win32)
-> hellow.c
1246 idents, 1235 lines, 48251 bytes, 0.001 s, 1234999 lines/s, 48.3 MB/s
Hello, world!
```

另外使用記憶體內編譯（64位元）方式時，方法與結果如下：

```
C:\tcc -v -bench -run hellow.c <enter>
tcc version 0.9.26 (x86-64 Win64)
-> hellow.c
1275 idents, 1234 lines, 48241 bytes, 0.001 s, 1234000 lines/s, 48.2 MB/s
Hello, world!
```

DLL程式編譯

以內附"Hello DLL"範例說明，該程式有兩個檔案dll.c與hello_dll.c，dll.c編譯完成後產生dll.dll，再以hello_dll.c來呼叫dll.dll dll.c程式碼如下

```
//+-------------------------------------------------------------------------
//
```

```
//  dll.c - Windows DLL example - dynamically linked part
//
#include <windows.h>
#define DLL_EXPORT __declspec(dllexport)

DLL_EXPORT void HelloWorld (void)
{
    MessageBox (0, "Hello World!", "From DLL", MB_ICONINFORMATION);
}
```

hello_dll.c程式碼如下

```
//+-------------------------------------------------------------------------
//
//  HELLO_DLL.C - Windows DLL example - main application part
//
#include <windows.h>

void HelloWorld (void);

int WINAPI WinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR     lpCmdLine,
    int       nCmdShow)
{
    HelloWorld();
    return 0;
}
```

編譯方式如下： 1. 首先以指令 -shared 來編譯dll.c檔案

```
C:\tcc -shared dll.c <enter>
```

產生dll.def與dll.dll 兩個檔案

2. 接著再以tiny_impdef.exe來import產生的dll.dll檔案。

```
C:\tiny_impdef dll.dll <enter>
```

3. 最後以主程式 hello_dll.c來連結dll.def檔案

```
C:\tcc hello_dll.c dll.def <enter>
```

產生hello_dll.exe檔案，執行hello_dll.exe檔即可看到結果。

新版本0.9.27的DLL編譯指令使用如下：

```
c:\tcc -shared dll.c
```

```
c:\tcc -impdef dll.dll
```

```
c:\tcc hello_dll.c dll.def
```

即可如上產生hello_dll.exe檔案。

編譯windows程式使用Win32 API

以內附"Hello WIN"範例說明

```
//+---------------------------------------------------------------------------
//
//  HELLO_WIN.C - Windows GUI 'Hello World!' Example
//
//+---------------------------------------------------------------------------

#include <windows.h>

#define APPNAME "HELLO_WIN"

char szAppName[] = APPNAME; // The name of this application
char szTitle[]   = APPNAME; // The title bar text
const char *pWindowText;

void CenterWindow(HWND hWnd);

//+---------------------------------------------------------------------------
//
//  Function:   WndProc
//
//  Synopsis:   very unusual type of function - gets called by system to
//              process windows messages.
//
//  Arguments:  same as always.
//----------------------------------------------------------------------------

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message) {

        // ----------------------- first and last
        case WM_CREATE:
            CenterWindow(hwnd);
            break;

        case WM_DESTROY:
            PostQuitMessage(0);
            break;

        // ----------------------- get out of it...
        case WM_RBUTTONUP:
            DestroyWindow(hwnd);
            break;

        case WM_KEYDOWN:
            if (VK_ESCAPE == wParam)
                DestroyWindow(hwnd);
            break;

        // ----------------------- display our minimal info
        case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC         hdc;
            RECT        rc;
            hdc = BeginPaint(hwnd, &ps);

            GetClientRect(hwnd, &rc);
            SetTextColor(hdc, RGB(240,240,96));
            SetBkMode(hdc, TRANSPARENT);
            DrawText(hdc, pWindowText, -1, &rc, DT_CENTER|DT_SINGLELINE|DT_VCENTER);

            EndPaint(hwnd, &ps);
            break;
        }

        // ----------------------- let windows do all other stuff
        default:
```

```
            return DefWindowProc(hwnd, message, wParam, lParam);
    }
    return 0;
}

//+--------------------------------------------------------------------------
//
//  Function:   WinMain
//
//  Synopsis:   standard entrypoint for GUI Win32 apps
//
//---------------------------------------------------------------------------
int APIENTRY WinMain(
        HINSTANCE hInstance,
        HINSTANCE hPrevInstance,
        LPSTR lpCmdLine,
        int nCmdShow
        )
{
    MSG msg;
    WNDCLASS wc;
    HWND hwnd;

    pWindowText = lpCmdLine[0] ? lpCmdLine : "Hello Windows!";

    // Fill in window class structure with parameters that describe
    // the main window.

    ZeroMemory(&wc, sizeof wc);
    wc.hInstance     = hInstance;
    wc.lpszClassName = szAppName;
    wc.lpfnWndProc   = (WNDPROC)WndProc;
    wc.style         = CS_DBLCLKS|CS_VREDRAW|CS_HREDRAW;
    wc.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
    wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor       = LoadCursor(NULL, IDC_ARROW);

    if (FALSE == RegisterClass(&wc))
        return 0;

    // create the browser
    hwnd = CreateWindow(
        szAppName,
        szTitle,
        WS_OVERLAPPEDWINDOW|WS_VISIBLE,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        360,//CW_USEDEFAULT,
        240,//CW_USEDEFAULT,
        0,
        0,
        hInstance,
        0);

    if (NULL == hwnd)
        return 0;

    // Main message loop:
    while (GetMessage(&msg, NULL, 0, 0) > 0) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return msg.wParam;
}

//+--------------------------------------------------------------------------

//+--------------------------------------------------------------------------

void CenterWindow(HWND hwnd_self)
{
    HWND hwnd_parent;
    RECT rw_self, rc_parent, rw_parent;
    int xpos, ypos;

    hwnd_parent = GetParent(hwnd_self);
    if (NULL == hwnd_parent)
```

```
        hwnd_parent = GetDesktopWindow();

    GetWindowRect(hwnd_parent, &rw_parent);
    GetClientRect(hwnd_parent, &rc_parent);
    GetWindowRect(hwnd_self, &rw_self);

    xpos = rw_parent.left + (rc_parent.right + rw_self.left - rw_self.right) / 2;
    ypos = rw_parent.top + (rc_parent.bottom + rw_self.top - rw_self.bottom) / 2;

    SetWindowPos(
        hwnd_self, NULL,
        xpos, ypos, 0, 0,
        SWP_NOSIZE|SWP_NOZORDER|SWP_NOACTIVATE
        );
}

//+---------------------------------------------------------------------------
```

編譯方式相同一般的C語言程式

```
C:\tcc hello_win.c
結果產生一hello_win.exe檔案
```

執行該hello_win.exe，即可看到以Win32 API編寫的Windows程式

使用組合語言

TinyCC即整合了Assembly於其中，使用TinyCC assembler的語法相容於GNU assembler即可，但是使用時仍是有限制條件如下：

- 必須是C或C++的指令有支援
- 由於指標符號相同於C，所以無法使用符號有"."或"$"
- 支援32位元為主
- 必須為inline assembler內嵌組合語言（或內聯彙編大陸用語）使用

相關支援的組合語言語法如下列所示：

```
.align n[,value]
.skip n[,value]
.space n[,value]
.byte value1[,...]
.word value1[,...]
.short value1[,...]
.int value1[,...]
.long value1[,...]
.quad immediate_value1[,...]
.globl symbol
.global symbol
.section section
.text
.data
.bss
.fill repeat[,size[,value]]
.org n
.previous
.string string[,...]
```

```
.asciz string[,...]
.ascii string[,...]
```

注意事項

為方便編譯，要將TCC原始碼內的libtcc.h於置於include內。