

Example

23

路徑、目錄與檔案 基本操作

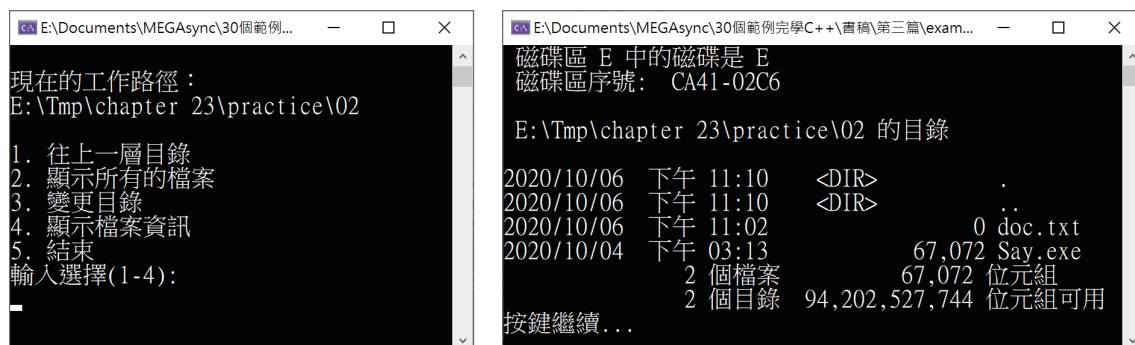
寫一程式，提供以下功能：往上一層目錄、顯示當前目錄裡的所有檔案、變更目錄、顯示所選定的檔案的大小。

一、學習目標

本範例學習 C++ 的路徑、目錄與檔案的基本操作。檔案放置在儲存媒體的特定地方，以路徑、目錄的組織方式提供檔案確切的位置；如此，才能對此檔案進行操作。

二、執行結果

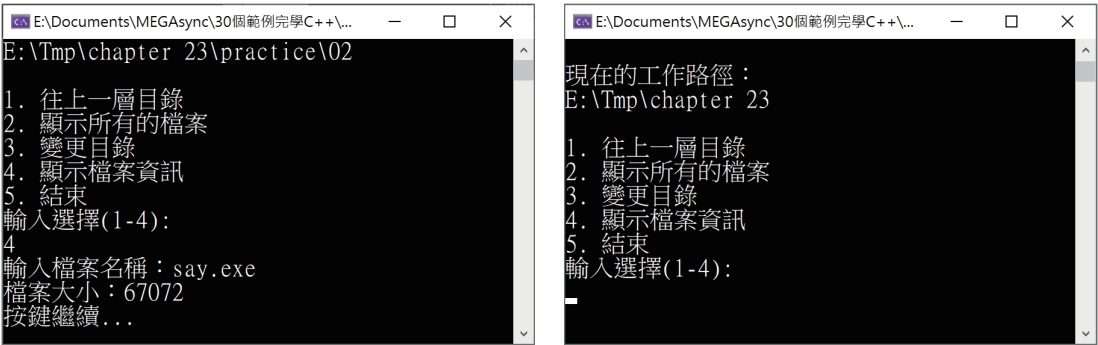
下圖左為初始畫面，假設目前執行檔案的路徑在 E:\Tmp\chapter 23\practice\02，則在最上方先顯示當前的工作路徑。下圖右為選擇「顯示所有的檔案」的畫面，此目錄下有 2 個檔案：doc.txt 與 Say.exe，並且可以看出 Say.exe 的檔案大小為 67072 位元組。



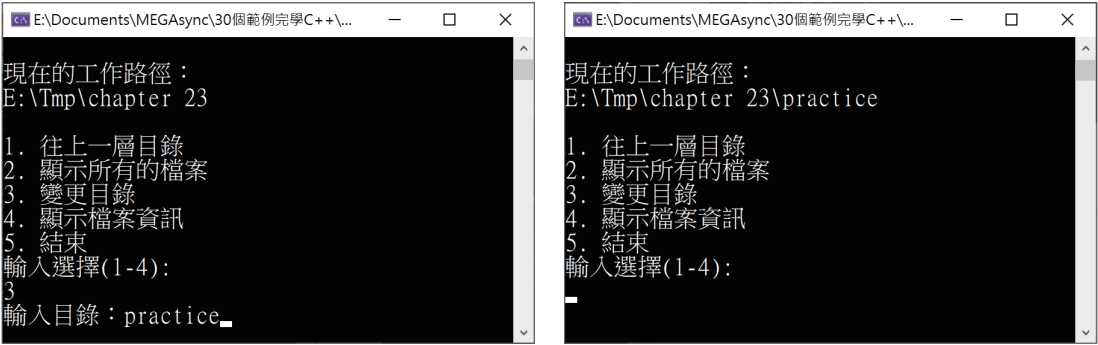
```
E:\Documents\MEGAsync\30個範例...  -  □  ×
現在的工作路徑：
E:\Tmp\chapter 23\practice\02
1. 往上一層目錄
2. 顯示所有的檔案
3. 變更目錄
4. 顯示檔案資訊
5. 結束
輸入選擇(1-4):
_

E:\Documents\MEGAsync\30個範例完學C++\書籍\第三篇\exam...  -  □  ×
磁碟區 E 中的磁碟是 E
磁碟區序號： CA41-02C6
E:\Tmp\chapter 23\practice\02 的目錄
2020/10/06 下午 11:10 <DIR>      .
2020/10/06 下午 11:10 <DIR>      ..
2020/10/06 下午 11:02              0 doc.txt
2020/10/04 下午 03:13          67,072 Say.exe
                2 個檔案          67,072 位元組
                2 個目錄    94,202,527,744 位元組可用
按鍵繼續...
```

下圖左為選擇了「顯示檔案資訊」的畫面：輸入了檔名 "say.exe" 之後顯示其檔案大小，此檔案的大小為 67072 位元組。下圖右為選擇了 2 次「往上一層目錄」之後的畫面，因此現在的工作路徑成為了：E:\Tmp\chapter 23。



接著選擇了「變更目錄」之後輸入要切換的目錄名稱："practice"，如下圖左所示。因此，目前的工作目錄變切換至：E:\Temp\chapter 23\prctice；如下圖右所示。



23-1 執行時期的錯誤代號與錯誤訊息

C++ 所提供的標準函式在執行時若發生錯誤，會將錯誤的代號與原因記錄下來。錯誤的代號記錄於 `errno` 與 `_doserrno` 全域變數，其中 `_doserrno` 特別針對輸出 / 輸入操作所發生的錯誤；`errno` 的錯誤代號說明如下表所列。

| 常數 | 值 | 說明 |
|--------|---|-----------|
| PERM | 1 | 不允許的操作。 |
| ENOENT | 2 | 無此檔案或目錄。 |
| ESRCH | 3 | 無此處理程序。 |
| EINTR | 4 | 中斷的函式。 |
| EIO | 5 | I/O 發生錯誤。 |
| ENXIO | 6 | 無此裝置或位址。 |

| | | |
|-----------|----|------------------------------|
| E2BIG | 7 | 引數太多。 |
| ENOEXEC | 8 | Exec 的格式錯誤。 |
| EBADF | 9 | 檔案號碼錯誤。 |
| ECHILD | 10 | 沒有衍生的處理程序。 |
| EAGAIN | 11 | 沒有處理程序、沒有足夠的記憶體，或已達最大達的巢狀層次。 |
| ENOMEM | 12 | 記憶體不足。 |
| EACCES | 13 | 無權限。 |
| EFAULT | 14 | 位址錯誤。 |
| EBUSY | 16 | 裝置或資源忙碌。 |
| EEXIST | 17 | 檔案已存在。 |
| EXDEV | 18 | 跨裝置連結。 |
| ENODEV | 19 | 無此裝置。 |
| ENOTDIR | 20 | 不是目錄。 |
| EISDIR | 21 | 是目錄。 |
| EINVAL | 22 | 無效的引數。 |
| ENFILE | 23 | 系統有太多已經開啓的檔案。 |
| EMFILE | 24 | 開啓太多檔案。 |
| ENOTTY | 25 | 不正確的 I/O 操作。 |
| EFBIG | 27 | 檔案太大。 |
| ENOSPC | 28 | 裝置的空間不夠。 |
| ESPIPE | 29 | 無效的搜尋。 |
| EROFS | 30 | 唯讀檔案系統。 |
| EMLINK | 31 | 太多連結。 |
| EPIPE | 32 | 管道中斷。 |
| EDOM | 33 | 數值引數超過定義域。 |
| ERANGE | 34 | 處理後的結果，其值超過值域。 |
| EDEADLK | 36 | 可能會發生資源死結。 |
| EDEADLOCK | 36 | 與 EDEADLK 相同。 |

| | | |
|--------------|----|-----------|
| ENAMETOOLONG | 38 | 檔案名稱太長。 |
| ENOLCK | 39 | 沒有鎖可用。 |
| ENOSYS | 40 | 不支援此函式。 |
| ENOTEMPTY | 41 | 不是空目錄。 |
| EILSEQ | 42 | 位元組序列不正確。 |
| STRUNCATE | 80 | 字串被截斷。 |

如下範例所示，函式 `_mkdir()` 會在目前的路徑下建立目錄 "subdir"，若目錄建立不成功則回傳 -1；當呼叫函式之後要立刻判斷 `errno` 的錯誤常數，以便做出相對應的處理。

```

1      :
2  if (_mkdir("subdir") == -1) // 建立 subdir 目錄發生錯誤
3      if (errno == EEXIST)    // 目錄已經存在
4      {
5          :
6      }
```

此外，可以透過呼叫 `_get_errno()` 函式取得錯誤代碼，以及呼叫 `strerror()` 函式取得錯誤的訊息。`strerror()` 函式的安全版本為 `strerror_s()`、`_strerror_s()`，寬字元版本為 `wcserror_s()`。

如下範例片段，欲在當前路徑之下建立目錄 "subdir"；但此目錄已經存在，所以會發生錯誤。程式碼第 1 行宣告 `error_t` 型別的變數 `err`，用於儲存錯誤代號。第 2 行宣告長度等於 100 的一維字元陣列 `errmsg`，用於儲存錯誤訊息。第 4 行呼叫 `_mkdir()` 函式建立目錄 `subdir`，若發生錯誤則執行第 5-12 行。

```

1  errno_t err;          // 儲存錯誤的代號
2  char errmsg[100];    // 儲存錯誤的訊息
3
4  if (_mkdir("subdir") == -1)
5  {
6      _get_errno(&err); // 取得錯誤的代號
7      _strerror_s(errmsg, 100, " 錯誤內容 "); // 取得錯誤訊息
8
9      cout << " 建立目錄 \"subdir\" 失敗，";
10     cout << " 錯誤代號：" << err << endl;
11     cout << errmsg << endl;
12 }
```

第 6 行呼叫函式 `_get_errno()` 取得錯誤代號，並傳入變數 `err` 的位址作為引數。第 7 行呼叫 `_strerror_s()` 取得錯誤訊息，並傳入 3 個引數，分別做為：儲存錯誤訊息的空間、儲存錯誤訊息的空間長度、顯示錯誤訊息的開頭字串。因此，第 9-11 行會顯示如下的訊息：

```
建立目錄 "subdir" 失敗，錯誤代號：17
錯誤內容：File exists
```

練習 1：顯示錯誤代號與訊息

使用 `_chdir()` 函式切換路徑至 `z:\temp\temp1`；顯示其發生錯誤的錯誤代號與訊息。

■ 解說

切換目錄的函式 `_chdir()` 需要傳入新的路徑字串作為引數，並引入 `direct.h` 標頭檔。若成功切換目錄則回傳 `0`，否則回傳 `-1`。

讀者的電腦不太可能有 `z:\temp\temp1` 這樣的路徑存在；因此會發生錯誤。所以必須判斷函式 `_chdir()` 的回傳值是否等於 `-1`；再使用 `_get_errno()` 與 `_strerror_s()` 函式取得錯誤代號與錯誤訊息。

■ 執行結果

因為沒有辦法切換路徑 `z:\temp\temp1`，所以錯誤代號等於 `2`；並且顯示錯誤訊息 `"No such file or directory"`。

```
錯誤代號：2
錯誤內容：No such file or directory
```

■ 程式碼列表

```
1 #include <iostream>
2 #include <direct.h>
3 using namespace std;
4
5 int main()
6 {
7     int ret;
8     errno_t err;
9     char errmsg[100];
10
11     ret=_chdir("z:\\temp\\temp1");
12
```

```

13     if (ret == -1)
14     {
15         _get_errno(&err);
16         _strerror_s(errmsg, 100, " 錯誤內容 ");
17
18         cout << " 錯誤代號：" << err << endl;
19         cout << errmsg << endl;
20     }
21
22     system("pause");
23 }

```

程式講解

1. 程式碼第 1-3 行引入 `iostream`、`direct.h` 標頭檔，與宣告使用 `std` 命名空間。
2. 程式碼第 7-9 行宣告變數。整數變數 `ret` 用於接收函式 `_chdir()` 的回傳值，`errno_t` 型別的變數 `err` 用於儲存錯誤代號，一維字元陣列 `errmsg` 則用於儲存錯誤訊息。
3. 程式碼第 11 行呼叫函式 `_chdir()` 切換路徑至 `z:\temp\temp1`。因為無此路徑，所以會發生錯誤，執行第 14-20 行的程式敘述。
4. 程式碼第 13 行判斷若切換路徑失敗，則執行第 14-20 行的程式敘述。第 15-16 行使用 `_get_errno()` 與 `_strerror_s()` 函式，分別取得錯誤代號與錯誤訊息，並儲存於變數 `err` 與 `errmsg`。第 18-19 行分別顯示錯誤代號與錯誤訊息。

23-2 路徑與目錄操作

目錄和路徑的組合指定了檔案在儲存媒體中的特定位置，例如以下 2 個路徑：

```

D:\doc\file\
e:\tmp\doc.txt

```

第 1 個路徑為指向硬碟 D 槽的目錄 `doc` 下的子目錄 `file`。第 2 個路徑指向硬碟 e 槽的目錄 `tmp` 裡的 1 個檔案 `doc.txt`。在 Windows 作業系統，不同層的路徑目錄使用分隔字元：反斜線 `"\"` 表示，英文字母不區分大小寫。路徑可以使用字串來表示；例如：

```

1 char strPath1[] = "d:\\doc\\file\\";
2 string strPath2 = "e:\\tmp\\do.txt\\";
3
4 cout << strPath1 << endl;
5 cout << strPath2 << endl;

```

程式碼第 1、2 行分別使用字元陣列和 `string` 型別宣告了字串 `strPath1` 與 `strPath2`，其字串的內容為路徑。使用字串表示的路徑時，其路徑分隔字元必須使用雙反斜線 `"\\"` 表示，但顯示的時候只會出現單個反斜線；如同第 4、5 行輸出路徑時，只會出現單個反斜線。

使用 C++ 標準函式

C++ 標準函式庫提供有關於目錄與路徑的相關函式，如下表所列。使用這些函式都需要引入 `direct.h` 或 `io.h` 標頭檔；若是寬字元版本則是需要引入 `wchar.h` 標頭檔。

| 函式 | 說明 |
|----------------------------|---|
| <code>access(a,b)</code> | 使用檢查模式 <code>b</code> 檢查目錄或文件 <code>a</code> 的狀態，檢查成功回傳 <code>0</code> 。若檔案或目錄不存在，或沒有指定檢查模式，則回傳 <code>-1</code> 。 <code>a</code> 為 <code>const char*</code> 型別， <code>b</code> 為整數型別。 |
| <code>chdir(a)</code> | 將目錄切換為 <code>a</code> ；切換成功則回傳 <code>0</code> ，否則回傳 <code>-1</code> 。 <code>a</code> 為 <code>const char*</code> 型別。 |
| <code>getcwd(a,b)</code> | 回傳目前的工作路徑，並儲存於字元指標 <code>a</code> 所指位址的空間， <code>b</code> 為欲取得的路徑長度；回傳 <code>char*</code> 型別。 <code>a</code> 為 <code>char*</code> 型別， <code>b</code> 為整數型別。 |
| <code>mkdir(a)</code> | 建立目錄 <code>a</code> ；建立成功則回傳 <code>0</code> ，否則回傳 <code>-1</code> 。 <code>a</code> 為 <code>const char*</code> 型別。 |
| <code>rmdir(a)</code> | 刪除目錄 <code>a</code> ；建立成功則回傳 <code>0</code> ，否則回傳 <code>-1</code> 。 <code>a</code> 為 <code>const char*</code> 型別。 |
| <code>_chdrive(a)</code> | 將磁碟機切換為 <code>a</code> ；切換成功則回傳 <code>0</code> ，否則回傳 <code>-1</code> 。 <code>a</code> 為整數型別， <code>1</code> 表示磁碟機 A， <code>2</code> 表示磁碟機 B，以此類推。 |
| <code>_getDrive()</code> | 取得目前磁碟機的代號，回傳值為整數型別。磁碟機 A 的代號為 <code>1</code> ，磁碟機 B 的代號為 <code>2</code> ，磁碟機 C 的代號為 <code>3</code> ，以此類推。 |
| <code>_getDrives()</code> | 以位元標示方式，回傳所有的磁碟機；回傳值為 <code>unsigned long</code> 型別。回傳值的第 <code>0</code> 個位元表示磁碟機 A，若此位元被標示為 <code>1</code> ，則表示有磁碟機 A；以此類推。 |
| <code>_get_pgmpr(a)</code> | 取得目前程式所在的路徑，並將路徑儲存於 <code>a</code> ；回傳值為 <code>error_t</code> 型別。 <code>a</code> 為 <code>char **</code> 型別。 |

函式 `access()` 原本用於檢查檔案的讀寫狀態，需要引入 `io.h` 標頭檔。如果搭配模式 `0` 使用時，則只會檢查檔案或是目錄是否存在；可使用的檢查模式如下所示。安全版本為與寬字元版本的函式分別為 `_access()` 與 `_waccess()`。

- 0 只檢查檔案或目錄是否存在
- 2 檢查檔案是否唯寫
- 4 檢查檔案是否唯讀
- 6 檢查檔案是否可讀可寫

函式 `chdir()` 用於切換目錄，安全版本為與寬字元版本的函式分別為 `_chdir()` 與 `_wchdir()`。在表達目錄時，通常以 `."` 表示目前的目錄，使用 `.."` 表示上一層的目錄，而 `"\\` 則表示根目錄。例如，若目前的工作路徑等於：`e:\tmp\document\music`，下列程式碼第 1 行切換到目前的路徑，所以工作路徑不會改變。第 2 行切換到上一層目錄，所以工作路徑為：`e:\tmp\document`。第 3 行切換路徑到根目錄，所以工作路徑為：`e:\`。

```
1 _chdir(".");
2 _chdir("..");
3 _chdir("\\");
```

函式 `mkdir()` 用於建立單層目錄；安全版本與寬字元版本的函式分別為 `_mkdir()` 與 `_wmkdir()`。若欲建立的目錄已經存在，則建立失敗並回傳 `-1`。例如：

```
1 _mkdir("e:\\dir1");           // 正確，建立單層目錄
2 _mkdir("e:\\dir2\\dir3");    // 錯誤，無法建立巢狀目錄
```

程式碼第 1 行在硬碟 `e` 槽下建立目錄 `dir1`。第 2 行要建立 2 層目錄：`dir2` 以及它的子目錄 `dir3`。由於函式 `mkdir()` 一次只能建立單層目錄；因此，第 2 行會建立失敗。

函式 `rmdir()` 用於刪除目錄，安全版本為與寬字元版本的函式分別為 `_rmdir()` 與 `_wrmdir()`。如果指定刪除的目錄裡面還有檔案或是子目錄，則會刪除失敗。

函式 `getcwd()` 可以取得目前的工作路徑，安全版本為與寬字元版本的函式分別為 `_getcwd()` 與 `_wgetcwd()`。函式 `getcwd()` 需要傳入 2 個引數，第 1 個引數可為字元指標或是字元型別的一維陣列，分別用於指向儲存工作路徑的空間的位址，或是作為儲存工作路徑的空間。第 2 個參數為所取得的路徑的長度。若所宣告的一維陣列容量不足以存放工作路徑，或是所設定的長度不足，則函式會回傳 `NULL`；如下範例。

```
1 char* curPath;
2 char arr[200];
3
4 curPath = _getcwd(arr, 200);
5
6 if(curPath!=NULL)
7     :
```

程式碼第 4 行呼叫 `_getcwd()` 取得目前的工作路徑，並將回傳的工作路徑儲存於字元指標 `curPath`，也可以儲存於陣列 `arr`。第 6 行判斷指標 `curPath` 若不等於 `NULL`，則表示正確地取得了工作路徑。以下是另一種更方便的使用方式：

```
curPath = _getcwd(NULL, 0);
```


將引數設為 `NULL` 與 `0`，函式 `_getcwd()` 會自動配置儲存路徑資料所需要的記憶體空間；因此，呼叫者只要接收回傳的字元指標即可。

函式 `_get_pgmprtr()` 取得的是目前正在執行的程式的路徑（包含程式名稱），與工作路徑不一定是相同的路徑。此函式其實是取得 C++ 全域變數 `_pgmprtr`（寬字元版為 `_wpgmprtr`）的內容；`_pgmprtr` 的內容就是目前正在執行的程式的路徑。但在 Visual Studio C++ 直接使用 `_pgmprtr` 會出現 C4996 的警告而發生錯誤；除了可以取消 C4996 的警告之外，也可使直接使用 `_get_pgmprtr()` 函式來取代 `_pgmprtr`；如下所示。

```

1 char* ptr=NULL;
2 string str;
3 _get_pgmprtr(&ptr);
4
5 cout << ptr << endl;
6 str = _pgmprtr; // 在 Visual Studio C++ IDE 中會出現 C4996 警告
7 cout << str << endl;
```

程式碼第 1 行宣告字元指標 `ptr`，用於儲存路徑字串。第 3 行呼叫 `_get_pgmprtr()` 函式，並傳入指標變數 `ptr` 的位址作為引數。第 6 行直接將 `_pgmprtr` 設定給字串 `str`，第 7 行顯示字串 `str` 的內容；第 5 行與第 7 行的內容會一樣：顯示目前正在執行的程式的路徑。

使用 `system()` 函式

在程式中操作路徑與目錄除了使用 C++ 所提供的函式之外，也能使用 `system()` 函式。`system()` 函式能使用「命令提示字元」視窗所使用的指令。例如，在本書中經常使用到的清除畫面：

```
system("cls");
```

字串 `"cls"` 就是在「命令提示字元」視窗所使用的指令，能夠清除「命令提示字元」視窗的畫面。諸如此類的指令很多，可以在「命令提示字元」視窗打入 `help` 指令查詢所有可以使用的指令。例如：

```

1 system("mkdir dir1");
2 system("dir");
3 system("rmdir dir1");
4 system("cd sub1");
5 system("cd ..");
6 system("cd \\");
```

程式碼第 1 行使用指令 "mkdir" 建立目錄 `dir1`，第 2 行使用指令 "dir" 顯示目前目錄下的所有檔案與目錄，第 3 行使用 "rmdir" 刪除目錄 `dir1`，第 4 行使用指令 "cd" 切換到目錄 `sub1`，第 5 行使用指令 "cd" 回到上一層到目錄 `..`，第 6 行使用指令 "cd" 回到根目錄 `\`。

練習 2：目錄與路徑的基本操作

先顯示當前的工作路徑，接著在目前的路徑下建立目錄 `subdir`。若建立目錄成功，則移動至目錄 `subdir` 並顯示工作路徑；然後再移動至上 2 層目錄，並顯示工作路徑。建立目錄前須先判斷目錄是否已經存在，並顯示錯誤訊息。建立目錄失敗也需顯示錯誤代號與錯誤訊息。

■ 解說

本練習會使用到的函式有：`_access()` 用於判斷目錄是否已經存在、`_mkdir()` 用於建立目錄、`_getcwd()` 用於取得目前的工作路徑、`_get_errno()` 用於取得錯誤代號、`strerror_s()` 用於取得錯誤訊息，與 `_chdir()` 用於改變路徑。

■ 執行結果

假設此程式位於路徑 `E:\Tmp\chapter 23\practice\02`，並且沒有 `subdir` 目錄。因此，執行程式之後發現沒有 `subdir` 目錄，便顯示錯誤代號 2，接著建立目錄 `subdir`。然後移動目錄到新建立的目錄，最後再往上移動 2 層目錄。

```
現在的工作路徑：E:\Tmp\chapter 23\practice\02
錯誤代號：2
已經建立 subdir 目錄
移動到目錄 subdir：
現在的工作路徑：E:\Tmp\chapter 23\practice\02\subdir
移動到上兩層目錄：
現在的工作路徑：E:\Tmp\chapter 23\practice
```

若再次執行此程式，由於目錄 `subdir` 已經建立；因此，工作路徑並不會改變，也會顯示 " 目錄已存在 "。

```
現在的工作路徑：E:\Tmp\chapter 23\practice\02
目錄已存在
```

程式碼列表

```

1  #include <iostream>
2  #include <direct.h>
3  #include <io.h>
4  #include <string.h>
5  using namespace std;
6
7  int main()
8  {
9      char* ptrPath;
10     errno_t err;
11
12     ptrPath = _getcwd(NULL, 0);
13     cout << " 現在的工作路徑：" << ptrPath << endl;
14
15     if (_access("subdir", 0) == -1) // 如果目錄 "subdir" 不存在
16     {
17         _get_errno(&err);
18         cout << " 錯誤代號：" << err << endl;
19         if (errno == ENOENT) // 無此檔案或是目錄
20         {
21             if (_mkdir("subdir") == -1) // 建立目錄 "subdir" 發生錯誤
22             {
23                 char errormsg[100];
24
25                 _get_errno(&err);
26                 _strerror_s(errormsg, 100, " 錯誤內容 ");
27
28                 cout << " 建立目錄 \"subdir\" 失敗，";
29                 cout << " 錯誤代號：" << err << endl;
30                 cout << errormsg << endl;
31                 exit(0);
32             }
33             else
34             {
35                 cout << " 已經建立 subdir 目錄 " << endl;
36
37                 cout << " 移動到目錄 subdir：" << endl;
38                 _chdir("subdir"); // 移動到剛建立的目錄 "subdir"
39                 ptrPath = _getcwd(NULL, 0);
40                 cout << " 現在的工作路徑：" << ptrPath << endl;

```

```

41
42         cout << " 移動到上兩層目錄：" << endl;
43         _chdir("../.."); // 移動到剛建立的目錄 "subdir"
44         ptrPath = _getcwd(NULL, 0);
45         cout << " 現在的工作路徑：" << ptrPath << endl;
46     }
47 }
48 }
49 else
50     cout << " 目錄已存在 " << endl;
51
52     system("pause");
53 }

```

程式講解

1. 程式碼第 1-5 行分別引入相關的標頭檔，與宣告使用 `std` 命名空間。
2. 程式碼第 9-10 行宣告變數。宣告字元指標變數 `ptrPath`，與 `errno_t` 型別的變數 `err`；分別用於儲存目前的工作路徑與錯誤代號。
3. 程式碼第 12 行取得目前的工作路徑，並儲存於變數 `ptrPath`。第 13 行顯示目前的工作路徑 `ptrPath`。
4. 第 15 行使用 `_access()` 函式判斷目錄 "subdir" 是否存在。若 "subdir" 目錄不存在則執行第 16-48 行；否則執行第 49-50 行，顯示 " 目錄已存在 "。
5. 程式碼第 17-18 行使用函式 `_get_errno()` 取得與顯示錯誤代號 `err`。第 19 行判斷若錯誤代號等於 `ENOENT`，則表示無 "subdir" 目錄，所以執行第 20-47 行：建立目錄、判斷是否發生錯誤、切換路徑。
6. 程式碼第 21 行使用 `_mkdir()` 函式建立目錄 "subdir"，若建立目錄失敗則執行第 22-32 敘述；若建立目錄成功則執行第 34-46 行程式敘述。

第 23 行宣告用於儲存錯誤訊息的變數 `errmsg`，第 25-26 行使用 `_get_errno()` 與 `_strerror_s()` 函式分別取得錯誤代號 `err` 與錯誤訊息 `errmsg`。第 28-30 行顯示錯誤代號與錯誤訊息。

7. 程式碼第 38-39 行使用函式 `_chdir()` 切換目錄至 "subdir"，並使用函式 `_getcwd()` 取的目前的工作路徑。因為已經建立目錄 `subdir`，所以可以順利切換路徑。

第 43-44 行使用函式 `_chdir()` 切換上 2 層目錄，並使用函式 `_getcwd()` 取的目前的工作路徑。若讀者執行此程式的工作路徑至少有 3 層，便可以順利切換至路徑到上 2 層。

23-3 檔案基本操作

檔案的基本操作包含了更改檔案名稱、移動檔案位置、刪除檔案、拷貝檔案與檔案屬性修改等。這些操作都能使用 C++ 的標準函式或是 `system()` 函式來處理。

使用 C++ 標準函式

與檔案相關的操作，都需要引入 `stdio.h` 或 `iostream` 標頭檔；若是寬字元版本則是需要引入 `wchar.h` 標頭檔。

| 函式 | 說明 |
|------------------------------|--|
| <code>chmod(a,b)</code> | 修改檔案 <code>a</code> 的存取權限為 <code>b</code> 。 <code>a</code> 為 <code>const char*</code> 型別， <code>b</code> 為整數型別。修改成功則回傳 <code>0</code> ，否則回傳 <code>-1</code> 。 |
| <code>chsize(a,b)</code> | 將已經開啓的檔案 <code>a</code> 的長度改為 <code>b</code> 。 <code>a</code> 為整數型別， <code>b</code> 為 <code>long</code> 型別。 |
| <code>copy(a,b)</code> | 將檔案 <code>a</code> 拷貝至檔案 <code>b</code> ；此為 C++ 17 版本的功能。 <code>a</code> 與 <code>b</code> 為 <code>const char*</code> 型別。 |
| <code>rename(a,b)</code> | 更改檔案或目錄 <code>a</code> 的名稱為 <code>b</code> ；更改成功則回傳 <code>0</code> ，否則回傳 <code>-1</code> 。 <code>a</code> 、 <code>b</code> 為 <code>const char*</code> 型別。 |
| <code>remove(a)</code> | 刪除檔案 <code>a</code> ；刪除成功則回傳 <code>0</code> ，否則回傳 <code>-1</code> 。 <code>a</code> 為 <code>const char*</code> 型別。 |
| <code>_findfirst(a,b)</code> | 尋找符合條件 <code>a</code> 的檔案，並儲存其檔案資訊於 <code>b</code> ；搜尋成功則回傳搜尋控制碼，否則回傳 <code>-1</code> 。回傳值為 <code>intptr_t</code> 型別， <code>a</code> 為 <code>const char*</code> 型別， <code>b</code> 為 <code>_finddata_t*</code> 結構型別。 |
| <code>_findnext(a,b)</code> | 使用 <code>_findfirst()</code> 函式所回傳的搜尋控制碼 <code>a</code> ，繼續搜尋下一個符合條件的檔案，並將檔案資訊儲存於 <code>b</code> 。 <code>a</code> 為 <code>intptr_t</code> 型別， <code>b</code> 為 <code>_finddata_t*</code> 結構型別。 |
| <code>_findclose(a)</code> | 結束由檔案搜尋控制碼 <code>a</code> 的檔案搜尋工作，並釋放相關資源；成功結束則回傳 <code>0</code> ，否則回傳 <code>-1</code> 。 <code>a</code> 為 <code>intptr_t</code> 型別。 |

函式 `chmod()` 用於設定檔案的存取權限，需要引入 `io.h` 標頭檔。安全版本為 `_chmod()`，寬字元版本為 `_wchmod()`，需引入 `wchar.h` 標頭檔；檔案的存取權限如下表所列。

► 檔案存取權限

檔案存取模式有 3 種：只能讀取、只能修改與能讀取與修改。

| 存取權限 | 說明 |
|-----------------------------------|----------|
| <code>_S_IREAD</code> | 只能讀取。 |
| <code>_S_IWRITE</code> | 只能修改。 |
| <code>_S_IREAD _S_IWRITE</code> | 能讀取也能修改。 |

當檔案權限設定為 `_S_IREAD` 時，檔案的存取權限設定為唯讀：只能開啓與讀取檔案，無法儲存修改的內容。若設定為 `_S_IWRITE` 則表示能寫入檔案。使用 `"|"` 運算子將 `_S_IREAD` 與 `_S_IWRITE` 結合：`_S_IREAD | _S_IWRITE`，則設定檔案能供讀取與寫入。

函式 `chsize()` 的安全版本為 `_chsize()`，用於修改檔案的大小。若調整之後的長度比原本檔案的長度小，則會刪除原來的檔案內容以符合調整後的長度；調整之後須將檔案關閉。

函式 `remove()` 用於刪除指定的檔案，指定的檔名可包含路徑；寬字元版為 `_wremove()`。函式 `rename()` 用於更改指定的檔案名稱，指定的檔名可包含路徑；寬字元版為 `_wrename()`。若新舊檔名的路徑不同，則表示移動檔案；新舊檔案所指的路徑必須已經存在。

函式 `_findfirst()` 的寬字元本為 `_wfindfirst()`，`_findnext()` 的寬字元本 `_wfindnext()`。 `_findfirst()`、`_findnext()` 與 `_findclose()` 此 3 個函式可用於搜尋符合條件的目錄與檔案。先使用函式 `_findfirst()` 搜尋指定路徑下的檔案或是目錄，當找到第 1 個符合的檔案或目錄之後，再使用 `while` 重複敘述搭配函式 `_findnext()` 繼續找其他符合條件的檔案或目錄；最後再使用 `_findclose()` 釋放所有的資源。其整個搜尋的程式架構如下所示：

```

1  intptr_t hFile = 0;
2  struct _finddata_t info;
3
4  if ((hFile = _findfirst("*", &info)) == -1) // 找不到檔案或是目錄
5  {
6      :
7  }
8  else
9  {
10     do
11     {
12         :
13     } while (_findnext(hFile, &info) == 0); // 繼續找檔案或是目錄
14 }
15 _findclose(hFile); // 釋放搜尋所使用的資源

```

程式碼第 1 行宣告 `intptr_t` 型別的變數 `hFile`；`intptr_t` 為整數指標的資料型別。第 2 行宣告 `_finddata_t` 結構型別的變數 `info`，用於儲存尋找到的檔案或目錄的資訊；`_finddata_t` 結構為 C++ 用於定義檔案或目錄資訊的結構。

第 4 行先使用 `_findfirst()` 函式尋找目前路徑下所有的檔案與目錄，若找不到任何資料則回傳 `-1`。`"*"` 表示要尋找所有的檔案與目錄；若要尋找特定的檔案類型，例如：尋找 `E:\tmp\` 路徑下所有的 `bmp` 類型的影像檔，則使用 `"E:\\tmp*.bmp"`。若找到第一個符合搜尋條件的檔案或目錄，則將回傳的搜尋控制碼儲存於變數 `hFile`，並執行第 9-14 行程式敘述。

程式碼第 10-13 行使用 `do...while` 重複敘述繼續尋找下一個符合搜尋條件的檔案或目錄；當已經無資料可尋找時，函式 `_findnext()` 則回傳 `-1`。若已經取得了檔案或是目錄的資訊，則可以使用這些資訊做其他的處理；例如，顯示檔案的檔名與大小：

```
1 cout << info.attrib << endl;
2 cout << info.size << endl;
```

► `_finddata_t` 結構

`_finddata_t` 結構用於描述檔案或是目錄的資訊，共有以下的結構成員：

| 成員名稱 | 資料型別 | 說明 |
|-------------------------------|-----------------------|---|
| <code>attrib</code> | <code>unsigned</code> | 檔案的屬性。 |
| <code>time_create</code> | <code>time_t</code> | 檔案的建立時間。 |
| <code>time_access</code> | <code>time_t</code> | 檔案最後的存取時間。 |
| <code>time_write</code> | <code>time_t</code> | 檔案最後一次被修改的時間。 |
| <code>size</code> | <code>_fsize_t</code> | 檔案的長度。 |
| <code>name[_MAX_FNAME]</code> | <code>char</code> | 檔案名稱。 <code>_MAX_FNAME</code> 被定義於 <code>stdlib.h</code> ，為檔名的最大長度。 |

其中，檔案屬性有以下的型態：

| 檔案屬性常數 | 數值 | 說明 |
|------------------------|----|------|
| <code>_A_NORMAL</code> | 0 | 一般檔案 |
| <code>_A_RDONLY</code> | 1 | 唯讀檔案 |
| <code>_A_HIDDEN</code> | 2 | 隱藏檔案 |
| <code>_A_SYSTEM</code> | 4 | 系統檔案 |
| <code>_A_SUBDIR</code> | 16 | 目錄 |
| <code>_A_ARCH</code> | 32 | 壓縮檔案 |

若要組合檔案屬性，則使用 `"|"` 運算子，例如：要設定檔案的屬性為唯讀與隱藏，則使用 `_A_RDONLY|_A_HIDDEN`。

使用 system() 函式

在程式中操作路徑與目錄除了使用 C++ 所提供的函式之外，也能使用 system() 函式。system() 函式能使用「命令提示字元」視窗所使用的指令。例如：

```
1 system("copy file1.txt file2.txt");
2 system("rename file1.txt newfile.txt");
3 system("del newfile.txt");
4 system("move file2.txt f:\\a");
5 system("attrib +r file1.txt");
```

程式碼第 1 行使用 "copy" 指令拷貝檔案 file1.txt 為 file2.txt。第 2 行使用指令 "rename" 將檔案 file1.txt 改名為 newfile.txt。第 3 行使用指令 "del" 刪除檔案 newfile.txt。第 4 行使用指令 "move" 將檔案 file2.txt 搬移至路徑 f:\a。第 5 行使用指令 "attrib" 將檔案 file1.txt 的屬性改為唯讀。

除此之外，尚有許多的指令可以使用；可以在「命令提示字元」視窗打入 help 指令查詢所有可以使用的指令。

練習 3：更改檔名與檔案屬性

先自行在執行程式的目錄裡建立文字檔 doc.txt。執行此程式之後，將檔案 doc.txt 更改檔名為 123.txt，接著再將此檔案的屬性更改為唯讀的狀態。

■ 解說

由於執行程式的所在路徑與其工作路徑可能會不同，所以首先要將程式的工作路徑更改為程式所在的路徑。接著使用函式 _access() 來判斷檔案 doc.txt 是否存在，才能繼續做後續的處理。更改檔名可以使用函式 rename()；更改檔案屬性為唯讀狀態，則可以使用函式 _chmod() 與檔案屬性常數 _A_RDONLY。

■ 執行結果

假設此程式的路徑為 E:\Tmp，並在此路徑下有檔案 doc.txt。更改檔名成功則顯示 "更改檔名成功"，更改其檔案屬性成功則顯示 "更改唯讀成功"。

```
目前的路徑：E:\Tmp\  
更改檔名成功  
更改唯讀成功
```


若找不到檔案 doc.txt，則顯示 " 檔案不存在 "。

目前的路徑：E:\Tmp\
檔案不存在

程式碼列表

```

1  #pragma warning(disable : 4996)
2  #include <iostream>
3  #include <direct.h>
4  #include <io.h>
5  #include <string>
6  using namespace std;
7
8  string getExePath()
9  {
10     string str;
11     int pos;
12
13     str = _pgmptr;
14     pos=str.find_last_of('\\');
15     if (pos == -1)
16         return str + "\\";
17     else
18         return str.substr(0, pos + 1);
19 }
20
21 int main()
22 {
23     string str;
24
25     str = getExePath();
26     cout << " 目前的路徑：" << str << endl;
27
28     if(_chdir(str.c_str())==-1)
29         cout << " 無法切換路徑 " << endl;
30     else
31     {
32         if (_access("doc.txt", 0) != -1)
33         {
34
35             if (rename("doc.txt", "123.txt") == 0)

```

```

36             cout << " 更改檔名成功 " << endl;
37         else
38             cout << " 更改檔名失敗 " << endl;
39
40         if (_chmod("123.txt", _S_IREAD) == 0)
41             cout << " 更改唯讀成功 " << endl;
42         else
43             cout << " 更改唯讀失敗 " << endl;
44     }
45     else
46         cout << " 檔案不存在 " << endl;
47 }
48 system("pause");
49 }

```

程式講解

1. 程式碼第 1 行宣告此程式要忽略 C4996 警告。第 2-6 行引入所需要的標頭檔與宣告使用 `std` 命名空間。
2. 程式碼第 8-19 行宣告自訂函式 `getExePath()`，此函式用於取得目前執行的程式的路徑。第 13 行使用全域變數 `_pgmptr` 取得執行程式的路徑，並儲存於字串變數 `str`。由於變數 `str` 中包含了路徑與檔案名稱，而此自訂函式只想取得路徑的部分；因此，第 14-18 行用於判斷與尋找變數 `str` 中最後的 1 個目錄與檔案的分隔字元 `'\'`；若找得到最後 1 個分隔字元，則回傳變數 `str` 中包含此分隔字元與其之前的部分（路徑的部分）。
3. 第 23 行宣告字元變數 `str`，第 25 行呼叫自訂函式 `getExePath()` 取得目前執行程式的路徑，並儲存於變數 `str`。第 28 行使用 `_chdir()` 函式切換目前的工作目錄至變數 `str`。
4. 若成功變換工作路徑，則程式碼第 32 行先使用 `_access()` 函式判斷檔案 `"doc.txt"` 是否存在，若檔案存在則執行第 33-44 行更改檔名與更改檔案屬性。
5. 程式碼第 35-38 行使用函式 `rename()` 將檔案 `doc.txt` 更改檔名為 `123.txt`，並判斷是否成功更改檔名。
6. 程式碼第 40-43 行使用函式 `_chmod()` 與檔案屬性常數 `_S_IREAD`，將檔案 `123.txt` 更改其檔案屬性為唯讀的狀態，並判斷是否更改成功。

三、範例程式解說

1. 建立專案，程式碼第 1-5 行引入所需要的標頭檔與宣告使用 `std` 命名空間。

```

1  #include <iostream>
2  #include <conio.h>
3  #include <direct.h>
4  #include <io.h>
5  using namespace std;

```

2. 程式碼第 8-15 行為自訂函式 `showCurrPath()`，此函式用於取得並顯示現在的工作路徑。第 10 行宣告字元指標 `ptrPath`，用於儲存現在的路徑。第 12 行使用 `_getcwd()` 函式取得現在的工作路徑，並儲存於變數 `ptrPath`。第 13-14 行顯示工作路徑 `ptrPath`。

```

8  void showCurrPath()
9  {
10     char* ptrPath;
11
12     ptrPath = _getcwd(NULL, 0);
13     cout << "\n\n 現在的工作路徑：" << endl;
14     cout << ptrPath << endl;
15 }

```

3. 程式碼第 18-27 行為自訂函式 `showMenu()`，用於顯示功能選單。

```

18 void showMenu()
19 {
20     cout << endl;
21     cout << "1. 往上一層目錄 " << endl;
22     cout << "2. 顯示所有的檔案 " << endl;
23     cout << "3. 變更目錄 " << endl;
24     cout << "4. 顯示檔案資訊 " << endl;
25     cout << "5. 結束 " << endl;
26     cout << "輸入選擇 (1-5): ";
27 }

```

4. 程式碼第 30-39 行為自訂函式 `showError()`，用於顯示錯誤代號與錯誤訊息。第 32 行宣告 `errno_t` 型別的變數 `err`，用於儲存錯誤代號。第 33 行宣告字元陣列 `errmsg`，用於儲存錯誤訊息。第 35-36 行分別使用函式 `_get_errno()` 與 `_strerror_s()` 取得錯誤代號與錯誤訊息，並儲存於變數 `err` 與 `errmsg`。第 37-38 行分別顯示錯誤代號與訊息。

```

30 void showError()
31 {
32     errno_t err;
33     char errmsg[100];
34
35     _get_errno(&err);
36     _strerror_s(errmsg, 100, " 錯誤內容 ");
37     cout << " 錯誤代號：" << err << endl;
38     cout << errmsg << endl;
39 }

```

5. 程式碼第 42-51 行為自訂函式 `chgDir()`，用於設定目錄。第 44 行宣告字串變數 `str`，用於儲存所輸入的目錄。第 46-47 行分別顯示提示訊息與讀取使用者所輸入的目錄，並將輸入的目錄儲存於變數 `str`。第 49-50 行使用函式 `_chdir()` 設定目錄，若設定目錄失敗則呼叫自訂函式 `showErr()` 顯示錯誤代號與訊息。

```

42 void chgDir()
43 {
44     string str;
45
46     cout << " 輸入目錄：";
47     cin >> str;
48
49     if (_chdir(str.c_str()) == -1)
50         showError();
51 }

```

6. 程式碼第 54-71 行為自訂函式 `showFileInfo()`，用於尋找指定的檔案並顯示此檔案的大小。第 56 行宣告 `intptr_t` 型別的變數 `hdfFile`，用於儲存函式 `_findfirst()` 回傳的搜尋控制碼。第 57 行宣告 `_finddata_t` 結構變數 `info`，用於儲存檔案的資訊。第 60-61 行顯示提示訊息，並將使用者所輸入的檔名儲存於字串變數 `str`。

第 63 行使用函式 `_findfirst()` 尋找指定的檔案 `str`，若找到此檔案則將檔案的資訊儲存於變數 `info`；若找不到此檔案則回傳 `-1`。第 67-68 行判斷若檔案屬性 `info.attrib` 不是目錄，則顯示檔案的大小 `info.size`。第 70 行使用 `_findclose()` 函式釋放搜尋所佔用的相關資源。

```

54 void showFileInfo()
55 {
56     intptr_t hFile = 0;
57     struct _finddata_t info;
58     string str;
59
60     cout << " 輸入檔案名稱：";
61     cin >> str;
62
63     if ((hFile = _findfirst(str.c_str(), &info)) == -1)
64         cout << " 沒有此檔案 " << endl;
65     else
66     {
67         if (info.attrib != _A_SUBDIR)
68             cout << " 檔案大小：" << info.size << endl;
69     }
70     _findclose(hFile);
71 }

```

7. 開始於 `main()` 主函式中撰寫程式。程式碼第 75 行宣告變數 `sel`，用於儲存使用者所輸入的選項。第 77-109 行為 `while` 無窮迴圈，第 79 行呼叫自訂函式 `showCurrPath()` 顯示現在的工作路徑，第 80 行呼叫自訂函式 `showMenu()` 顯示選單；第 81 行讀取使用者所輸入的選項，並儲存於變數 `sel`。

```

75 int sel;
76
77 while (true)
78 {
79     showCurrPath(); // 顯示工作路徑
80     showMenu();     // 顯示選單
81     cin >> sel;

```

8. 程式碼第 83-105 行為 `switch..case` 選擇敘述；根據變數 `sel` 執行相關的功能。第 85-88 行為「往上一層目錄」的功能，第 86-87 行使用 `_chdir()` 函式切換至上一層目錄，若切換目錄失敗則呼叫自訂函式 `showError()` 顯示錯誤代號與訊息。

第 90-91 行為「顯示所有的檔案」的功能，使用函式 `system()` 以及指令 `"dir"` 顯示目前目錄裡面所有的檔案與目錄。第 93-94 為「變更目錄」的功能，呼叫自訂函式 `chgDir()` 改變目錄。

```

83     switch (sel)
84     {
85         case 1:
86             if (_chdir("..") == -1) // 往上一層目錄
87                 showError();
88             break;
89
90         case 2: system("dir");      // 顯示所有檔案
91             break;
92
93         case 3: chgDir();          // 改變目錄
94             break;

```

第 96-97 為「顯示檔案資訊」的功能，呼叫自訂函式 `showFileInfo()` 顯示檔案的大小。程式碼第 99-100 行為「結束」的功能，呼叫函式 `exit()` 結束程式。第 102-104 行為 `default` 區塊；若輸入錯誤的選項則第 103 行顯示錯誤訊息。第 107-108 行等待使用者按任一鍵後繼續執行。

```

96         case 4: showFileInfo();    // 顯示檔案大小
97             break;
98
99         case 5: exit(0);
100             break;
101
102         default:
103             cout << " 輸入錯誤, ";
104             break;
105     }
106
107     cout << " 按鍵繼續 ...";
108     while (!_kbhit());
109 }
110
111 system("pause");

```

重點整理

1. 目錄與檔案可能因其存取權限受到限制，所以無法存取。因此，在變更目錄或是讀取檔案資訊時，都需要檢查是否變更或是讀取成功。也有可能檔案或目錄不存在，而發生錯誤。
2. 目錄與檔案的基本操作都有 2 種方式：使用 C++ 所提供的函式，或是使用函式 `system()` 加上指令。

程式碼列表

```

1  #include <iostream>
2  #include <conio.h>
3  #include <direct.h>
4  #include <io.h>
5  using namespace std;
6
7  //----- 顯示工作路徑 -----
8  void showCurrPath()
9  {
10     char* ptrPath;
11
12     ptrPath = _getcwd(NULL, 0);
13     cout << "\n\n 現在的工作路徑：" << endl;
14     cout << ptrPath << endl;
15 }
16
17 //----- 顯示選單 -----
18 void showMenu()
19 {
20     cout << endl;
21     cout << "1. 往上一層目錄 " << endl;
22     cout << "2. 顯示所有的檔案 " << endl;
23     cout << "3. 變更目錄 " << endl;
24     cout << "4. 顯示檔案資訊 " << endl;
25     cout << "5. 結束 " << endl;
26     cout << "輸入選擇 (1-5): ";
27 }
28
29 //----- 顯示錯誤代號與訊息 -----
30 void showError()
31 {
32     errno_t err;
33     char errmsg[100];
34
35     _get_errno(&err);
36     _strerror_s(errmsg, 100, " 錯誤內容 ");
37     cout << " 錯誤代號：" << err << endl;
38     cout << errmsg << endl;
39 }
40

```

```
41 //----- 改變目錄 -----
42 void chgDir()
43 {
44     string str;
45
46     cout << " 輸入目錄 : ";
47     cin >> str;
48
49     if (_chdir(str.c_str()) == -1)
50         showError();
51 }
52
53 //----- 顯示檔案大小 -----
54 void showFileInfo()
55 {
56     intptr_t hFile = 0;
57     struct _finddata_t info;
58     string str;
59
60     cout << " 輸入檔案名稱 : ";
61     cin >> str;
62
63     if ((hFile = _findfirst(str.c_str(), &info)) == -1)
64         cout << " 沒有此檔案 " << endl;
65     else
66     {
67         if (info.attrib != _A_SUBDIR)
68             cout << " 檔案大小 : " << info.size << endl;
69     }
70     _findclose(hFile);
71 }
72
73 int main()
74 {
75     int sel;
76
77     while (true)
78     {
79         showCurrPath(); // 顯示工作路徑
80         showMenu();     // 顯示選單
81         cin >> sel;
82     }
```



```

83         switch (sel)
84         {
85             case 1:
86                 if (_chdir("..") == -1) // 往上一層目錄
87                     showError();
88                 break;
89
90             case 2: system("dir");      // 顯示所有檔案
91                 break;
92
93             case 3: chgDir();           // 改變目錄
94                 break;
95
96             case 4: showFileInfo();     // 顯示檔案大小
97                 break;
98
99             case 5: exit(0);
100                break;
101
102             default:
103                 cout << " 輸入錯誤，";
104                 break;
105         }
106
107         cout << " 按鍵繼續 ...";
108         while (!_kbhit());
109     }
110
111     system("pause");
112 }

```

本章習題

1. 使用 `_mkdir()` 函式建立目錄 `subdir`；若發生錯誤，則顯示其發生錯誤的錯誤代號與錯誤訊息。
2. 在目前的工作路徑的上一層，建立目錄 `subdir1`，接著在目錄 `subdir1` 之下再建立目錄 `subdir2`。
3. 刪除第 2 題所建立的 2 個目錄。
4. 使用函式 `_findfirst()` 與 `_findnext()` 列出目前目錄下的所有檔案。

