<p.13>

原為:

\* void 指標永遠不會和其他指標相等，然而，兩個指派為 NULL 的 void 指標則會相等。

改為:

\* void 指標永遠不會和其他指標相等，然而，兩個指派為 NULL 的 void 指標則會相等，void 指標的實際行為會依系統而有所不同。

<p.13>

原為:

• void 指標與char 指標有相同的表示方式與記憶體對齊（memory alignemtn）。

改為:

• void 指標與char 指標有相同的表示方式與記憶體對齊（memory alignment）。

<p.23>

原為:

warning: pointer of type 'void \*' used in arithmetic [-Wpointerarith]

改為:

warning: pointer of type 'void \*' used in arithmetic [-Wpointer-arith]

<p.23>

原為:

運算的結果會讓 pv 指標增加 4 位元組的位址。

改為:

運算的結果會讓 pv 指標增加 1 位元組的位址，大多數編譯器將 void 資料型別視為只有一個位元組的大小，因此對於 void 指標，位址會增加 1 位元組。

大多數編譯器會對 void 資料型別的 sizeof 運算發出警告，這與一般將 void 指標的大小視為 4 位元組不同。

<p.24>

原為:

p0 112

p0 116

p0 120

改為

p0 112

p1 116

p2 120

<p.46>

原為:

char\* buffer[size];

改為:

char buffer[size];

<p.78>第一段

原為:

因此，範例陣列中的所有元素都會初始化為 NULL：

改為:

因此，範例陣列中剩下的其他元素都會初始化為 NULL：

<p.87>

原為：

與 vector+i 產生的機器碼不同

改為：

與 \*(vector+i) 產生的機器碼不同

原為:

vector+i 表示法產生的機器碼

改為:

\*(vector+i) 產生的機器碼

<p.116> 圖 505 中全域記憶體區塊裡

原為:

globalHeader

改為

displayHeader

<p.123>

原為:

char\* path = "C:";

char\* currentPath = (char\*) malloc(strlen(path)+2);

currentPath = strcat(currentPath, "\\");

改為:

char \*path = "C:";

char\* currentPath = (char\*) malloc(strlen(path)+2);

strcpy(currentPath, path);

currentPath = strcat(currentPath, "\\");

<p.124>

原為:

否則，會解讀成遞增字串解參考後指向的字元，與程式預期的行為不同：

改為:

由於後制遞增運算子（postfix-operator ++)的優先權比解參考運算子還高，小括號並非絕對必要，但加上小括號能更明確的表明程式設計師的意圖：

<p.125>

原為:

這種做法除了多餘又不必要之外，也常會產生警告訊息

改為:

雖然程式可以執行，但實際傳入的是字元指標的指標而非字元指標，這會產生警告訊息。

<p.133>

原為:

typedef int(fptrOperation)(const char\*, const char\*);

改為:

typedef int(\*fptrOperation)(const char\*, const char\*);

<p.140>

原為:

當宣告這個類型的變數，或是動態配置這個類型的記憶體時，結構中的三個指標只是沒有意義的資料。

改為:

當宣告這個類型的變數，或是動態配置這個類型的記憶體時，結構中的三個指標只是沒有意義的資料，然而，若是宣告為 static 變數（例如宣告在函式之外的全域變數或是在函式當中加上 static 關鍵字），那麼結構成員就會被初始化為 0（指標則是初始化為 NULL）。

<p.143> 圖 6-4

原為:

person

改為:

ptrPerson

<p.155>

原為:

void insertNode(TreeNode \*\*root, COMPARE compare, void\* data) {

TreeNode \*node = (TreeNode\*) malloc(sizeof(TreeNode));

node->data = data;

node->left = NULL;

node->right = NULL;

if (\*root == NULL) {

\*root = node;

return;

}

while (1) {

if (compare((\*root)->data, data) > 0) {

if ((\*root)->left != NULL) {

\*root = (\*root)->left;

} else {

(\*root)->left = node;

break;

}

} else {

if ((\*root)->right != NULL) {

\*root = (\*root)->right;

} else {

(\*root)->right = node;

break;

}

}

}

}

改為:

void insertNode(TreeNode \*\*realRoot, COMPARE compare, void\* data) {

TreeNode \*node = (TreeNode\*) malloc(sizeof (TreeNode));

node->data = data;

node->left = NULL;

node->right = NULL;

TreeNode \*root = \*realRoot;

if (root == NULL) {

\*realRoot = node;

return;

}

while (1) {

if (compare((root)->data, data) > 0) {

if ((root)->left != NULL) {

root = (root)->left;

} else {

(root)->left = node;

break;

}

} else {

if ((root)->right != NULL) {

root = (root)->right;

} else {

(root)->right = node;

break;

}

}

}

}