

使用 Visual Studio 偵錯器

H

And so shall I catch the fly.

—William Shakespeare

*We are built to make mistakes,
coded for error.*

—Lewis Thomas

*What we anticipate seldom
occurs; what we least expect
generally happens.*

—Benjamin Disraeli

He can run but he can't hide.

—Joe Louis

*It is one thing to show a man that
he is in error, and another to put
him in possession of truth.*

—John Locke

學習目標

在本章中，你將學到：

- 在偵錯器中設定中斷點以及執行程式。
- 使用 Continue 指令繼續執行。
- 使用 Locals 視窗檢視和修改變數的值。
- 使用 Watch 視窗計算運算式的值。
- 使用 Step Into、Step Out、Step Over 指令控制執行流程。
- 使用 Autos 視窗檢視附近的敘述式使用的變數。



本章綱要

H.1 簡介

H.2 中斷點和 Continue 指令

H.3 Locals 和 Watch 視窗

H.4 使用 Step Into、Step Over、Step Out、Continue 指令控制執行流程

H.5 Autos 視窗

H.6 總結

摘要 | 術語 | 自我測驗 | 自我測驗解答

H.1 簡介

你在第 2 章學到兩種錯誤：編譯錯誤和邏輯錯誤，也學到如何從你的程式碼排除編譯錯誤。邏輯錯誤（也稱為 **bugs**）並不會讓程式無法編譯，但是會讓程式在執行的時候產生錯誤的結果。大部分的 C++ 編譯器廠商會提供一種稱為**偵錯器 (debugger)** 的軟體，讓你可以監視程式的執行，找到邏輯錯誤並排除。偵錯器將是你最重要的程式開發工具之一。這篇附錄示範 Visual Studio 偵錯器的主要功能，附錄 I 會討論 GNU C++ 偵錯器的特性和功能。

H.2 中斷點和 Continue 指令

我們從研究中斷點 (**breakpoint**) 開始學習偵錯器，這是可以設定在任一行執行碼的標記。當程式執行時遇到中斷點會暫停，讓你可以檢查變數的值，有助於判斷是否有邏輯錯誤。例如，你可以檢查儲存計算結果的變數的值，判斷計算是否執行正確。請注意，在不能執行一行程式碼（例如註解）設定中斷點的話，實際上的中斷點是設定在這個函式下一行可執行的程式碼。

我們使用圖 H.3 的程式解釋偵錯器的這項特性，這個程式建立並操作 Account 類別（圖 H.1 到圖 H.2）的物件。程式從 main（圖 H.3，第 10–27 行）開始執行，第 12 行建立一個 Account 物件，初始餘額是\$50.00。Account 的建構子（圖 H.2，第 9–21 行）接收一個引數，指定 Account 的初始 balance。圖 H.3 的第 15 行使用 Account 的成員函式 getBalance 印出初始的帳戶餘額。第 17 行宣告一個區域變數 withdrawalAmount，儲存從使用者讀取的提款金額。第 19 行提示使用者輸入提款金額，第 20 行把金額輸入 withdrawalAmount。第 23 行使用 Account 的成員函式 debit，從 balance 扣除提款金額。最後，第 26 顯示新的 balance。

```

1 // Fig. H.1: Account.h
2 // Definition of Account class.
3 class Account
4 {
5 public:
6     Account( int ); // constructor initializes balance
7     void credit( int ); // add an amount to the account balance
8     void debit( int ); // subtract an amount from the account balance
9     int getBalance(); // return the account balance
10 private:
11     int balance; // data member that stores the balance
12 }; // end class Account

```

圖 H.1 Account 類別的標頭檔

```

1 // Fig. H.2: Account.cpp
2 // Member-function definitions for class Account.
3 #include <iostream>
4 using namespace std;
5
6 #include "Account.h" // include definition of class Account
7
8 // Account constructor initializes data member balance
9 Account::Account( int initialBalance )
10 {
11     balance = 0; // assume that the balance begins at 0
12
13     // if initialBalance is greater than 0, set this value as the
14     // balance of the account; otherwise, balance remains 0
15     if ( initialBalance > 0 )
16         balance = initialBalance;
17
18     // if initialBalance is negative, print error message
19     if ( initialBalance < 0 )
20         cout << "Error: Initial balance cannot be negative.\n" << endl;
21 } // end Account constructor
22
23 // credit (add) an amount to the account balance
24 void Account::credit( int amount )
25 {
26     balance = balance + amount; // add amount to balance
27 } // end function credit
28
29 // debit (subtract) an amount from the account balance
30 void Account::debit( int amount )
31 {
32     if ( amount <= balance ) // debit amount does not exceed balance
33         balance = balance - amount;
34     else // debit amount exceeds balance
35         cout << "Debit amount exceeded account balance.\n" << endl;
36 } // end function debit

```

圖 H.2 Account 類別的定義

H-4 C++程式設計藝術(第七版)(國際版)

```
37
38 // return the account balance
39 int Account::getBalance()
40 {
41     return balance; // gives the value of balance to the calling function
42 } // end function getBalance
```

圖 H.2 Account 類別的定義 (續)

```
1 // Fig. H.3: figL_03.cpp
2 // Create and manipulate Account objects.
3 #include <iostream>
4 using namespace std;
5
6 // include definition of class Account from Account.h
7 #include "Account.h"
8
9 // function main begins program execution
10 int main()
11 {
12     Account account1( 50 ); // create Account object
13
14     // display initial balance of each object
15     cout << "account1 balance: $" << account1.getBalance() << endl;
16
17     int withdrawalAmount; // stores withdrawal amount read from user
18
19     cout << "\nEnter withdrawal amount for account1: "; // prompt
20     cin >> withdrawalAmount; // obtain user input
21     cout << "\nattempting to subtract " << withdrawalAmount
22         << " from account1 balance\n\n";
23     account1.debit( withdrawalAmount ); // try to subtract from account1
24
25     // display balances
26     cout << "account1 balance: $" << account1.getBalance() << endl;
27 } // end main
```

圖 H.3 要偵錯的測試類別

在 Visual C++ 2008 Express 中建立專案

利用下列步驟，你可以使用圖 H.1-H.3 的程式碼建立一個專案。

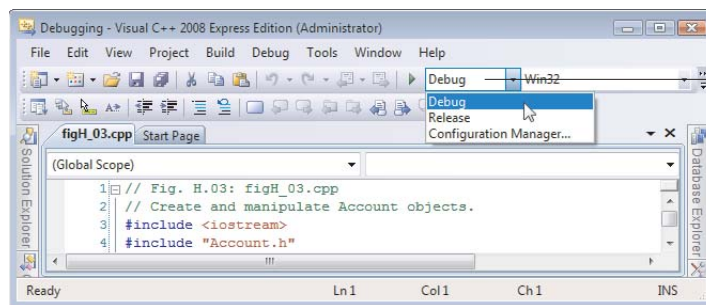
1. 在 Visual C++ 2008 Express 中選擇 **File > New > Project...**，顯示 **New Project** 對話框。
2. 在 **Project types:** 中選擇 **Win32**；在 **Templates** 中，選擇 **Win32 Console Application**。

3. 在 **Name:** 欄位輸入專案名稱，在 **Location:** 欄位，填入你想將專案儲存在電腦的哪個位置，然後按下 **OK**。
4. 在 **Win32 Application Wizard** 對話框，按下 **Next >**。
5. 在 **Application type:** 中選擇 **Console application**；在 **Additional options:** 下選擇 **Empty project**，接著按下 **Finish**。
6. 在 Visual C++ 的 **Solution Explorer** 視窗，對著專案的 **Source Files** 資料夾點選右鍵，然後選擇 **Add > Existing Item...**，以顯示 **Add Existing Item** 對話框。
7. 找到附錄 H 範例程式碼的資料夾，選擇全部三個檔案，接著點選 **Add**。

進入偵錯模式，插入中斷點

依照下列步驟，你可以使用中斷點和各種偵錯器指令，檢視圖 H.3 宣告的變數 `withdrawalAmount` 的值。

1. **啟動偵錯器。** 偵錯器預設是啟動，如果沒有的話，必須要修改工具列的 **Solution Configurations 組合方塊** (圖 H.4) 設定，請按一下組合方塊的向下箭頭，然後選擇 **Debug**。



Solution Configurations
組合方塊

圖 H.4 啟動偵錯器

2. **插入中斷點。** 在 **Solution Explorer** 中雙按 `figH_03.cpp` 檔案將它開啓。要插入中斷點的話，在想要中斷的程式碼旁邊的**邊界指令列 (margin indicator bar)** (圖 H.5 的程式碼視窗左邊的灰色邊界) 按一下，或是在該行程式碼按右鍵選擇 **Breakpoint > Insert Breakpoint**。你可以視需要加任意多個中斷點。在程式碼的第 17 行和第 21 行設定中斷點。在你按的邊界指令列上會出現紅色的實心圓，表示已經設定一個中斷點 (圖 H.5)。當程式執行時，偵錯器在會任何有中斷

H-6 C++程式設計藝術(第七版)(國際版)

點的那行暫停程式執行。當偵錯器暫停程式執行時，程式稱之為處在**中斷模式 (break mode)**。你可以在程式執行之前設定中斷點，也可以在程式執行到中斷模式時設定中斷點。

3. **啟動偵錯器**。在程式編輯器設定中斷點之後，請選擇 **Build > Build Solution** 編譯程式，然後選擇 **Debug > Start Debugging** 開始偵錯流程。[請注意：假如你沒有先編譯程式，當你選擇 **Debug > Start Debugging** 時，程式還是會先被編譯。] 假如你在執行的是主控台應用程式 (console application)，會出現命令提示字元 (Command Prompt) 視窗 (圖 H.6)，可以和程式互動 (輸入和輸出)。當執行到第 17 行的中斷點時，偵錯器會進入中斷模式。
4. **檢查程式的執行**。程式在第一個中斷點 (第 17 行) 進入中斷模式，IDE 會變成主要視窗(圖 H.7)。第 17 行左邊的**黃色箭頭 (yellow arrow)** 表示下一行要執行的敘述式。

中斷點
邊界指示列
中斷點

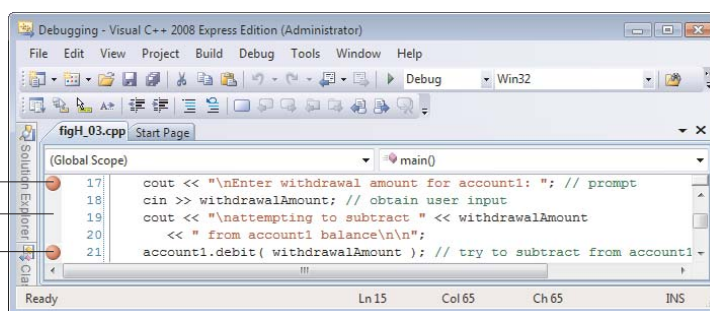


圖 H.5 設定二個中斷點

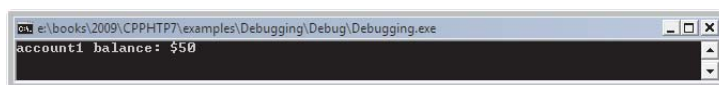


圖 H.6 Inventory 程式執行時

黃色箭頭表示
下一個要執行
的敘述式

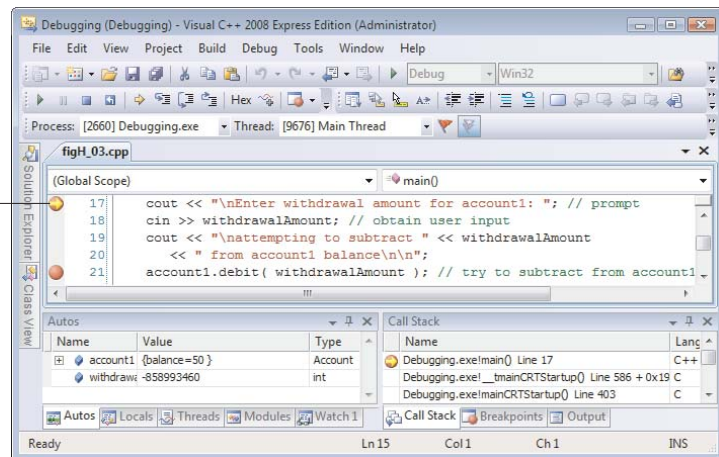


圖 H.7 程式在第一個中斷點暫停執行

5. 使用 **Continue** 指令繼續執行。要繼續執行的話，請選擇 **Debug > Continue**。Continue 指令會執行下一個執行敘述式直到遇到下一個中斷點，或是直接到 main 結尾，看哪一個先遇到。程式會繼續執行，然後在第 18 行等待輸入。輸入 13 當作提款金額。程式會繼續執行，直到第 21 行的中斷點。請注意，當你把滑鼠游標移到變數名稱 `withdrawalAmount` 之上，這個變數儲存的值會顯示在 **Quick Info** 方塊 (圖 H.8)。你隨後就會知道，這可以幫助你找到程式的邏輯錯誤。

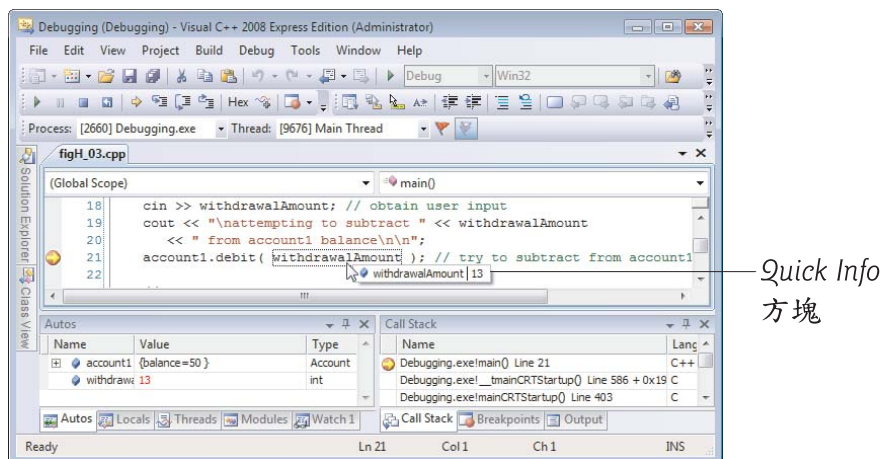
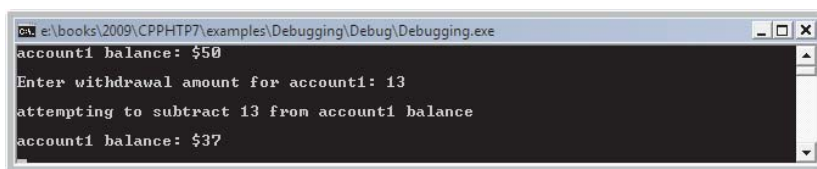


圖 H.8 Quick Info 方塊，顯示變數的值

H-8 C++程式設計藝術(第七版)(國際版)

6. 在 **main** 的結束括弧設定中斷點。在第 25 行左側的邊界指示列按一下，在第 25 行的原始碼設定一個中斷點。這可以避免程式在顯示結果之後立刻關閉。當沒有其他的中斷點暫停執行時，程式會執行到結束，然後命令提示字元視窗會關閉。如果你沒有設定這個中斷點，就無法在主控台視窗關閉之前，看到程式的輸出。
7. 繼續執行程式。使用 **Debug > Continue** 指令，繼續執行下一個中斷點之前的程式。程式會顯示計算的結果 (圖 H.9)。



```
Ca: e:\books\2009\CPPHTP7\examples\Debugging\Debug\Debugging.exe
account1 balance: $50
Enter withdrawal amount for account1: 13
attempting to subtract 13 from account1 balance
account1 balance: $37
```

圖 H.9 程式的輸出

8. 取消中斷點。在邊界指示列的中斷點按一下。
9. 結束程式執行。選擇 **Debug > Continue** 繼續執行程式直到結束。

你在這一節學到如何啓用偵錯器和設定中斷點，可以讓你在程式執行時檢查程式碼的結果。你也學到在程式遇到中斷點暫停執行後，如何繼續執行，以及如何取消中斷點。

H.3 Locals 和 Watch 視窗

在前一節有學到 Quick Info 的功能，可以让你檢查變數的值。你在這一節會學習如何使用 **Locals** 視窗，在程式執行時替變數設定新的值。你也會使用 **watch** 視窗檢查更複雜的運算式的值。

1. 插入中斷點。清除目前的中斷點。在第 21 行左側的邊界指示列按一下，在第 21 行的原始碼設定一個中斷點 (圖 H.10)。在程式碼第 24 行左邊的邊界指令列按一下，在第 24 行設定一個中斷點。

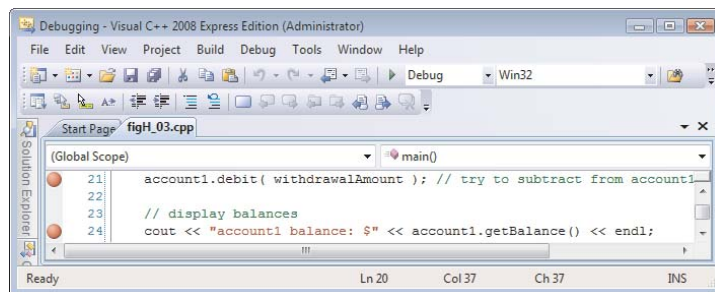


圖 H.10 在第 25 和 28 行設定中斷點

2. **開始偵錯。**選擇 **Debug > Start**。在 **Enter withdrawal amount for account1**：然後按 Enter，讓程式讀取你剛剛輸入的數值。程式會執行到第 21 行的中斷點。
3. **暫停程式執行。**當偵錯器到達第 21 行時，會進入中斷模式（圖 H.11）。此時，第 18 行的敘述式已經把你的輸入 (13) 存入 withdrawalAmount，第 19–20 行輸出說明程式將要提款，而第 21 行是下一個要執行的敘述式。

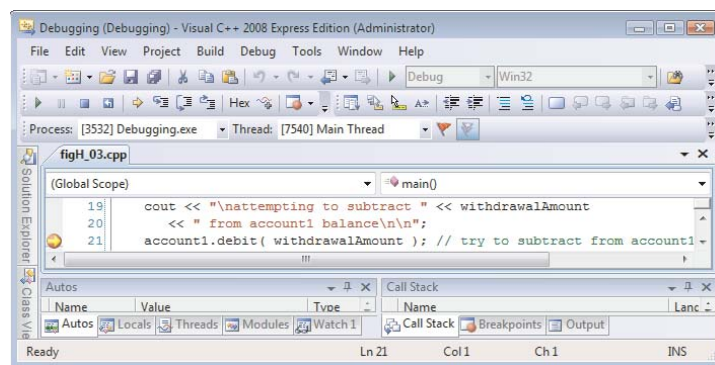


圖 H.11 當偵錯器到達第 25 行的中斷點，程式會暫停執行

4. **檢查資料。**一旦程式進入中斷模式，你可以使用偵錯器的 **Locals** 視窗瀏覽你的區域變數，該視窗通常位於 IDE 的最底端。假如沒有出現，請選擇 **Debug > Windows > Locals**。圖 H.12 會顯示 main 的區域變數值：account1 和 withdrawalAmount (13)。

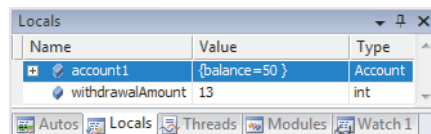


圖 H.12 檢查變數 withdrawalAmount

5. **求算術和布林運算式的值。**你可以使用其中一個 **Watch** 視窗求算術和布林運算式的值。你最多可以顯示 4 個 Watch 視窗。選擇 **Debug > Windows > Watch > Watch 1**。在 **Name** 那行的第一列，輸入 $(withdrawalAmount + 3) * 5$ ，然後按輸入鍵。請注意此時 **Value** 那一列會顯示運算式的值（這裡是 80）（圖

H-10 C++程式設計藝術(第七版)(國際版)

H.13)。在 Name 的下一列，輸入 `withdrawalAmount == 3`，然後按輸入鍵。此運算式用來判斷 `withdrawalAmount` 是否等於 3。包含 `==` 符號的運算式 (或是其他關係或等號運算子)，視為布林運算式。運算式傳回的值是 `false` (圖 H.13)，因為 `withdrawalAmount` 目前的值是 13，不是數值 3。

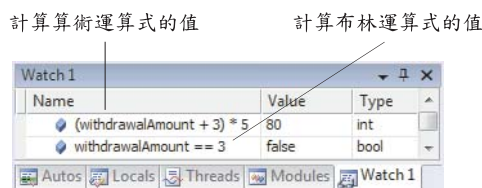
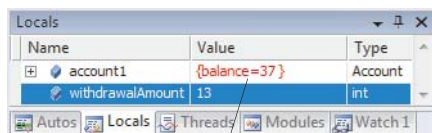


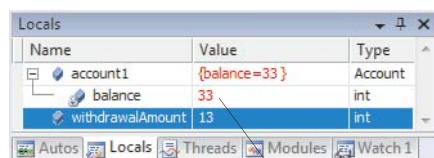
圖 H.13 檢查運算式的值

6. **繼續執行。**請選擇 **Debug > Continue** 繼續執行。第 21 行執行會從帳戶提出指定的金額，然後程式再停在第 24 行。請選擇 **Debug > Windows > Locals** 或是在 Visual Studio 的底端按下 **Locals** 頁籤以顯示 **Locals** 視窗。`account1` 中更新的 `balance` 值以紅色顯示，表示自從上次的中斷點之後已經更動 (圖 H.14)。在 **Locals** 視窗的 **Name** 那行，按一下 `account1` 左側的加號方塊，可以看到 `account1` 的每個資料成員的值。當物件有許多資料成員時，這個功能是很有用的。
7. **修改數值。**根據使用者輸入的值 (13)，程式輸出的帳戶餘額應該是 \$37。然而，你可以在程式執行時期，使用 **Locals** 視窗修改變數的值。這是很很有用的，可以實驗不同的數值，找到程式的邏輯錯誤。在 **Locals** 視窗，按一下 `balance` 那一列的數值欄選取數值 37。鍵入 33 然後按下 **Enter** 鍵。偵錯器會修改 `balance` 的值，然後新的值以紅色顯示 (圖 H.15)。



account1 資料成員 `balance` 的數值顯示成紅色

圖 H.14 顯示區域變數的值



在 Locals 視窗中修改的變數

圖 H.15 修改變數的值

8. 在 **main** 的結束括弧設定中斷點。在第 25 行設定一個中斷點，避免程式在顯示結果之後立刻關閉。如果你沒有設定這個中斷點，就無法在主控制台視窗關閉之前，看到程式的輸出。
9. 觀察程式結果。請選擇 **Debug > Continue** 繼續執行程式。main 函式執行到第 29 行的 **return** 的敘述式為止，並且顯示結果。請注意結果是 \$33 (圖 H.16)。這顯示步驟 7 把 **balance** 的值，從計算的值 (37) 改成 33。

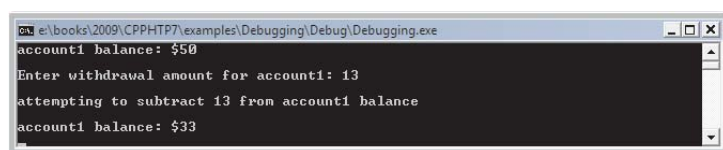


圖 H.16 在修改變數 account1 之後顯示的輸出

10. 停止偵錯。請選擇 **Debug > Stop Debugging**，會關閉命令提示字元視窗，取消其餘的中斷點。

你在這一節學到如何使用偵錯器的 **Watch** 和 **Locals** 視窗，求算術和布林運算式的值。你也學到如何在程式執行時，修改變數的值。

H.4 使用 Step Into、Step Over、Step Out、Continue 指令控制執行流程

有時候你會需要逐行執行程式，才可以確認程式碼正確執行，或找到邏輯錯誤並修正。在這一節學的這些指令讓你可以在逐行執行一個函式、一次執行一個函式的所有敘述式、或是只執行一個函式剩餘的敘述式 (如果你已經執行這個函式的一些敘述式)。

H-12 C++程式設計藝術(第七版)(國際版)

1. **設定中斷點。**在邊界指示列按一下，在第 21 行設定中斷點。
2. **啟動偵錯器**選擇 **Debug > Start**。在 **Enter withdrawal amount for account1:**提示之後輸入 13。當執行到第 21 行的中斷點，程式會暫停。
3. **使用 Step Into 指令。****Step Into** 指令會執行程式下一個要執行的敘述式 (第 21 行)，然後馬上暫停。如果 **Step Into** 指令要執行的敘述式是一個函式呼叫，控制權會轉移到被呼叫的函式。**Step Into** 指令讓你可以進入一個函式，逐一執行這個函式的每個敘述式，確定執行無誤。請選擇 **Debug > Step Into** 進入 **debit** 函式。再選一次 **Debug > Step Into** 讓黃色箭號指向 **Account.cpp** 的第 31 行，如圖 H.17 所示。

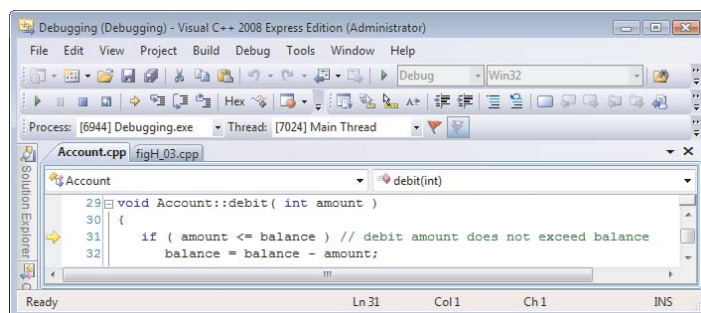


圖 H.17 走進 debit 函式

4. **使用 Step Over 指令。**請選擇 **Debug > Step Over** 執行目前的敘述式 (圖 H.17，第 31 行)，然後控制權轉移到第 32 行 (圖 H.18)。當下一個要執行的敘述式不包含函式呼叫時，**Step Over** 指令就和 **Step Into** 指令一樣。你會在步驟 10 看到 **Step Over** 指令和 **Step Into** 指令的差異。

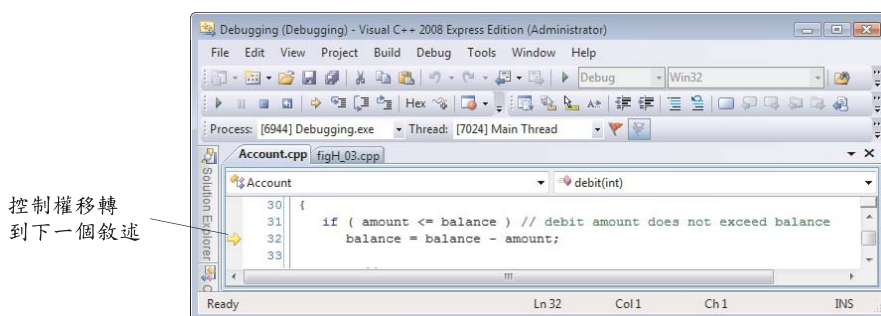


圖 H.18 走過 debit 函式的一個敘述式

5. **使用 Step Out 指令。**選擇 **Debug > Step Out** 執行函式剩餘的敘述式，然後控制權返回到包含這個函式呼叫的下一個可執行敘述式 (圖 H.3，第 28 行)。在一些比較長的函式，你會想看幾行關鍵的程式碼，然後繼續偵錯呼叫者的程式碼。這種情況下 **Step Out** 指令很有用，不需要逐行執行整個函式。
6. **設定中斷點。**在圖 H.3 的第 25 行，main 的末端設定一個中斷點，你會在下一個步驟使用這個中斷點。
7. **使用 Continue 指令。**選擇 **Debug > Continue** 執行到下一個在第 25 行的中斷點。當你不想要逐行執行許多行程式碼直到下個中斷點，這個功能可以省時。
8. **停止偵錯器。**選擇 **Debug > Stop Debugging** 結束偵錯，會關閉命令提示字元視窗，
9. **啟動偵錯器。**在我們示範下一個偵錯器功能之前，你必須要再次啟動偵錯器。如同步驟 2 一樣再次啟動，然後在提示訊息後面輸入 13。當偵錯器到達第 21 行時，會進入中斷模式。
10. **使用 Step Over 指令。**選擇 **Debug > Step Over** (圖 H.19)。還記得當下一個要執行的敘述式不包含函式呼叫時，這個指令就和 **Step Into** 指令一樣。如果下一個要執行的敘述式包含函式呼叫，被呼叫的函式會整個執行 (不會停留在函式內的任何敘述式)，然後黃色箭頭前進到目前函式下一個可執行的那行 (在函式呼叫之後)。目前偵錯器要執行 main 的第 21 行 (圖 H.3)，第 21 行呼叫 debit 函式，然後偵錯器會在第 24 行暫停執行，這是目前函式 (main) 下一行可執行的程式。

當選擇 Step Over 指令時 debit 函式會整個執行

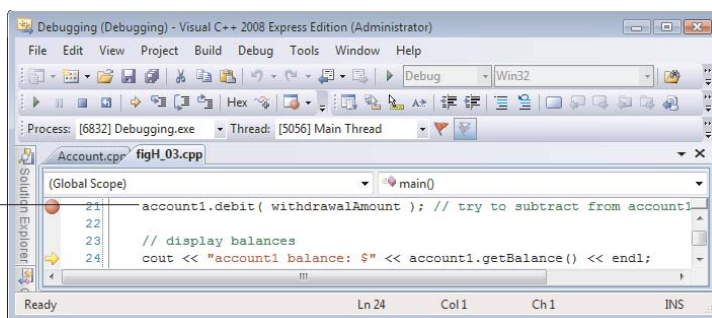


圖 H.19 使用偵錯器的 Step over 指令

11. **停止偵錯器。**請選擇 **Debug > Stop Debugging**，會關閉命令提示字元視窗，取消其餘的中斷點。

H-14 C++程式設計藝術(第七版)(國際版)

在這一節，你學到如何使用偵錯器的 **Step Into** 指令，在程式執行時進行呼叫函式偵錯。你看到如何使用 **Step Over** 指令，經過函式呼叫。使用 **Step Out** 指令繼續執行，直到目前函式結束。你也學到使用 **Continue** 指令，繼續執行程式直到下一個中斷點，或是程式結束。

H.5 Autos 視窗

Autos 視窗可以顯示前一個執行的敘述式使用的變數（包括函式回傳值），以及下一個要執行的敘述式使用的變數。

1. **設定中斷點**。在邊界指示列按一下，在 main 函式的第 10 和 18 行設定中斷點。
2. **使用 Autos 視窗**。選擇 **Debug > Start**，啟動偵錯器。當偵錯器在第 10 行進入中斷模式時，選擇 **Debug > Windows > Autos** 打開 **Autos** 視窗（圖 H.20）。因為你剛開始執行程式，**Autos** 視窗只會列出接下來要執行的敘述式的變數，也就是 account1 物件的值和型別。觀察物件內儲存的值，可以讓你驗證程式正確處理這些變數。請注意 account1 內含一個很大的負數。每次程式執行時這個值可能都不一樣，這是 account1 的未初始值。這個值是無法預測的（而且通常是不想要的），正說明 C++ 變數在使用前要初始化的重要性。

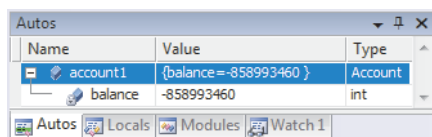


圖 H.20 Autos 視窗顯示 account1 物件的狀態

3. **使用 Step Over 指令**。選擇 **Debug > Step Over** 執行第 10 行。在 account1 初始化後，**Autos** 視窗會更新其資料成員 balance 的值（圖 H.21）。balance 的值以紅色顯示表示有變化。

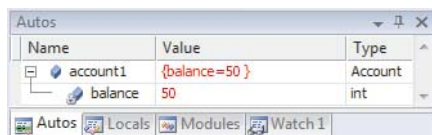
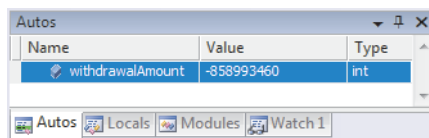
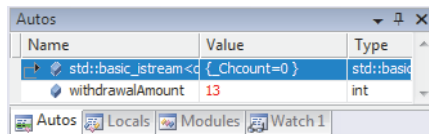


圖 H.21 Autos 視窗顯示 account1 物件在初始化後的狀態

4. **繼續執行**。選擇 **Debug > Continue**，繼續執行程式，直到第 18 行設定的第二個中斷點。**Autos** 視窗（圖 H.22）顯示未初始化的區域變數 `withdrawalAmount`，是一個很大的負數。

圖 H.22 Autos 視窗顯示區域變數 `withdrawalAmount`

5. **輸入資料**。選擇 **Debug > Step Over** 執行第 18 行。在程式的輸入提示輸入提款金額。**Autos** 視窗（圖 H.23）的區域變數 `withdrawalAmount` 的值，會更新成你輸入的數值。

圖 H.23 Autos 視窗顯示更新後的區域變數 `withdrawalAmount`

6. **停止偵錯器**。選擇 **Debug > Stop Debugging** 結束偵錯，取消其餘的中斷點。

H.6 總結

在本附錄中，你學會如何在 Visual Studio 偵錯器中插入、停用、取消中斷點。中斷點可以暫停程式執行，讓你檢視變數的值。幫助你找到程式的邏輯錯誤並修正它。你也學會如何使用 **Locals** 和 **Watch** 視窗，檢視運算式的值，並改變變數的值。你也學會如何使用偵錯器指令 **Step Into**、**Step Over**、**Step Out** 和 **Continue** 判斷函式是否正確執行。最後，你學會了如何 **Autos** 視窗觀察上一個和下一個敘述式中的變數內容。

摘要

H.1 簡介

- 大部分的 C++ 編譯器廠商會提供一種稱為偵錯器的軟體，讓你可以監視程式的執行，找到邏輯錯誤並排除。
- 中斷點是一個標示，可以設在任一行可執行的程式碼。當程式執行時遇到中斷點，就會暫停執行。
- 偵錯器預設是啟動，如果沒有的話，必須要修改 Solution Configurations 組合方塊的設定。

H.2 中斷點和 Continue 指令

- 想要插入中斷點，可以按一下這行程式碼旁邊的邊界指示列，或是在這行程式碼按右鍵選擇 Breakpoint > Insert Breakpoint。你按的地方會出現一個紅色的實心圓，表示已經設定一個中斷點。
- 當程式開始執行時，會停在有中斷點的地方。這稱為處在中斷模式。
- 有一個黃色箭頭表示這行有下一個要執行的敘述式。
- 當你把滑鼠游標移到變數名稱之上，這個變數儲存的值會顯示在 Quick Info 方塊。
- 想要停用一個中斷點，請在設定中斷點的那行程式碼按右鍵，然後選擇 Breakpoint > Disable Breakpoint。被停用的中斷點以紅色空心圓表示。
- 想要取消不再需要的中斷點的話，請在中斷點設定的那行按右鍵，然後選擇 Breakpoint > Delete Breakpoint。你也可以在邊界指示列的圓圈按一下，取消這個中斷點。

H.3 Locals 和 Watch 視窗

- 一旦程式進入中斷模式，你可以使用偵錯器的 Locals 視窗瀏覽你的變數。想要檢視 Locals 視窗，請選擇 Debug > Windows > Locals。
- 你可以使用其中一個 Watch 視窗求算術和布林運算式的值。
- 更新的變數會以紅色顯示，表示自從上次的中斷點之後，他們的值已有改變。
- 在 Locals 視窗的 Name 那行按一下物件旁的加號方塊，可以讓你觀察物件的每個資料成員。
- 你可以按一下變數的 Value 欄位，修改他在 Locals 視窗內的值。

H.4 使用 Step Into、Step Over、Step Out、Continue 指令控制執行流程

- Step Into 指令會執行程式下一個要執行的敘述式 (以黃色標示)。如果下一個敘述式要執行函式呼叫，而且你選擇 Step Into，控制權會轉到被呼叫的函式。

- 當下一個要執行的敘述式不包含函式呼叫時，Step Over 指令就和 Step Into 指令一樣。如果下一個要執行的敘述式包含函式呼叫，會執行整個被呼叫函式，然後黃色箭頭會移到目前函式的下一行可執行程式碼。
- Step Over 指令會執行這個函式剩餘的敘述式，然後把控制權傳回呼叫這個函式的地方。
- Continue 指令會執行下一個執行敘述式直到遇到下一個中斷點，或是直接到 main 結尾，看哪一個先遇到。

H.5 Autos 視窗

- Autos 視窗讓你可以觀察上一個執行的敘述式使用的變數內容，Autos 視窗也會列出下一個要執行的敘述式的值。

術語

Autos 視窗 (Autos window)	Solution Configurations 組合方塊
中斷模式 (break mode)	(Solution Configurations combo box)
中斷點 (breakpoint)	Step Into 指令 (Step Into command)
錯誤 (bug)	Step Out 指令 (Step Out command)
Continue 指令 (Continue command)	Step Over 指令 (Step Over command)
偵錯器 (debugger)	Watch 視窗 (Watch window)
Locals 視窗 (Locals window)	黃色箭號 (yellow arrow)
邊界指示列 (margin indicator bar)	中斷模式的黃色箭頭 (yellow arrow in break mode)
Quick Info 方塊 (Quick Info box)	

自我測驗

H.1 填充題：

- 當偵錯器在中斷點暫停程式執行時，程式稱之處在_____模式。
- Visual Studio 2005 的_____功能，可以讓你將滑鼠移到程式的變數名稱上，觀察變數的值。
- 你可以用偵錯器的_____視窗，檢查運算式的值。
- 當下一個要執行的敘述式不包含函式呼叫時，_____指令就和 Step Into 指令一樣。

H.2 說明下列何者為對，何者為錯。如是錯的，請解釋為什麼。

- 當程式到達中斷點而暫停執行時，下一個要執行的敘述式就是中斷點後的這一個。
- 當變數的值改變時，在 Autos 和 Locals 視窗會變成黃色。

H-18 C++程式設計藝術(第七版)(國際版)

- c) 在偵錯過程中，Step Out 指令會執行目前函式剩餘的敘述式，然後傳回控制權到呼叫這個函式的地方。

自我測驗解答

H.1 a)中斷。 b) Quick Info 方塊。 c) Watch。 d) Step Over。

H.2 a) 錯。當程式到達中斷點而暫停執行時，下一個要執行的敘述式是中斷點所在的這一個。

- b) 錯。當變數改變時會變成紅色。c) 對。